KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Continual Learning in Neural Networks

Rahaf Aljundi

Supervisor:
Prof. dr. ir. T. Tuytelaars

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

September 2019

Continual Learning in Neural Networks

Rahaf ALJUNDI

Examination committee:
Prof. dr. ir. H. Neuckermans, chair
Prof. dr. ir. T. Tuytelaars, supervisor
Prof. dr. ir. L. Van Gool
Prof. dr. ir. H. Van Hamme
Prof. dr. ir. R. Vogels
Prof. dr. A. Vedaldi
(University of Oxford)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

September 2019

Acknowledgements

Obtaining a PhD with impactful research was a target that I set 4 years ago. Working towards this target was not an easy task at all. Without the support of many people, this wouldn't have been possible.

I would like to express my deepest appreciation to my supervisor prof. Tinne Tuytelaars for her continual support during the course of my PhD. I am especially thankful for the freedom and the trust that I was given to pursue the research I am passionate about. I wouldn't have been able to proceed without your guidance.

I am also thankful to prof. Matthew Blaschko for the great mathematical discussions. I would like to thank my PhD committee prof. Van Gool, prof. Van Hamme, prof. Vogels, prof. Vedaldi and prof. Neuckermans for their careful reading of my PhD manuscript, the fruitful discussions, and insights.

My research was funded by FWO whose vision inspired me to explore continual learning, my beloved topic. I would like to thank them for the trust and I wish my work met their expectations.

During my PhD I have collaborated with many awesome researchers. Jay Chakravarty, with him I have got my first two papers. Amal, with her I started a crazy rush towards a deadline in two months and we won a paper and a friendship. Francisca who joined the lab for a short period but we shared best moments of hard work and laugh. I am really thankful for getting to know you. Klaas, thanks for making our last CVPR paper realizable.

I would like to express my thanks to Marcus Rohrbach and Mohammed Elhosieny for the fruitful collaboration and the great discussions on Continual Learning. During my last research visit to Mila, I had the chance to work with prof. Yoshua Bengio, Min Lin, Eugene Belilovsky, Massimo Caccia and Lucas Caccia. It was the shortest period in which I got to learn the most and survive the freezing Montreal.

Working towards a PhD can't be done without sharing the best and tough moments with the friends and colleges of Visics and without the help of the friendly staff (Annitta,

Bert, Patricia and Paul). Special mention to Bert, Davy and Ali with whom I spent an unforgettable week in Hawaii. Jose, I would like to thank you for the expert answers whenever I needed and for the TITAN X that made my experiments achievable.

This thesis is dedicated to my parents who planted the seeds of science in me, whose circumstances didn't support their research path. I hope I have achieved your dream. My parents, brothers, and sister, without your support I wouldn't have the chance to pursue my career.

Finally, my biggest lovely thanks go to my husband, Mostafa, who stood by me in each moment of this endless deadlines journey, listened to my complaints, my worries and never stopped putting confidence in me.

Abstract

Artificial neural networks have exceeded human level performance in accomplishing several individual tasks (e.g. voice recognition, object recognition, and video games). However, such success remains modest compared to human intelligence that can learn and perform an unlimited number of tasks. Humans ability of learning and accumulating knowledge over their lifetime is an essential aspect of their intelligence. In this respect, continual machine learning aims at a higher level of machine intelligence through providing the artificial agents with the ability to learn online from a non-stationary and never-ending stream of data.

A key component of such a never-ending learning process is to overcome the catastrophic forgetting of previously seen data, a problem that neural networks are well known to suffer from. The work described in this thesis has been dedicated to the investigation of continual learning and solutions to mitigate the forgetting phenomena in neural networks.

To approach the continual learning problem, we first assume a task incremental setting where tasks are received one at a time and data from previous tasks are not stored. We start by developing a system that aims for an expert level performance on each learned task. It reserves a separate specialist model for each task and sequentially learns a gate to forward the input data to the corresponding specialist. We then consider the incremental learning of multiple tasks using a shared model of fixed capacity. For each task, we identify the most informative features and minimize their divergence during the learning of later tasks; using as a proxy the current task data.

As an alternative to relying on the current task data, which might be of a very different distribution than previous data, important parameters in a model can be identified and future changes on them get penalized. However, when accounting for an unlimited sequence of tasks, it is impossible to preserve all the previous knowledge. As an adaptive method to specific test conditions, we propose to learn the important parameters at deployment time while the model is active in its test environment. As a result, catastrophic forgetting is overcome but graceful selective forgetting is tolerated.

To further account for future tasks, we study the role of sparsity in continual learning. We propose a new regularizer that significantly reduces the percentage of parameters dedicated to each task and as a consequence remarkably improves the continual learning performance.

Since the task incremental setting can't be assumed in all continual learning scenarios, we also study the more general online continual setting. We consider an infinite stream of data drawn from a non-stationary distribution with a supervisory or self-supervisory training signal. We first propose a protocol to bring our work on regularizing the important parameters to the online continual learning setting and show an improved learning performance over different streams of data. As to account for more challenging situations where the input distribution is experiencing bigger changes, we explore the use of a fixed buffer of samples selected from the previous history. We propose a sample selection method that makes no assumption on the data generating distribution. To the best of our knowledge, we were the first to tackle the online continual learning problem.

The proposed methods in this thesis have tackled important aspects of continual learning. They were evaluated on different benchmarks and over various learning sequences. Advances in the state of the art of continual learning have been shown and challenges for bringing continual learning into application were critically identified.

Beknopte samenvatting

Artificiële neurale netwerken scoren voor vele individuele taken (bv. Spraakherkenning, objectherkenning en videospellen) beter dan de mens. Het succes blijft echter beperkt, als we kijken naar het oneindig aantal taken dat een mens kan leren en uitvoeren. De mogelijkheid om een leven lang te leren en kennis te blijven vergaren, is een essentiël typering van menselijke intelligentie. Met dit in gedachte mikt continu leren op een hoger niveau van machinale intelligentie, door de artificiële intelligentie de mogelijkheid te bieden om on-line te blijven leren met een oneindige stroom aan data. Een cruciale component in zo'n nooit eindigend leerproces is het overwinnen van het rampzalige vergeten van eerder verkregen data, een bekend probleem bij neurale netwerken. Het werk dat beschreven wordt in deze thesis is gewijd aan het onderzoek naar continu leren en het oplossen van het fenomeen van vergeten in neurale netwerken.

Bij de aanpak van het continu leren probleem, gaan we uit van een stapsgewijze methode waar de taken na elkaar volgen en data van de vorige taak niet bewaard blijft. We starten met de ontwikkeling van een systeem dat tot doel heeft om voor elke taak het expert niveau te benaderen. Het reserveert een apart specialistisch model voor elke taak en leert achtereenvolgens om via een poort de juiste input door te geven aan de corresponderende specialist. Daarna bekijken we het stapgewijs leren van meerdere taken door gebruik te maken van een gedeeld model met een vastgelegde capaciteit. Voor elke taak identificeren we de meest informatieve kenmerken en minimaliseren we de divergentie tijdens het leren van latere taken; met als proxy de data voor de huidige taak.

Als een alternatief voor het zich baseren op de data voor de lopende taak, die misschien een heel andere distributie heeft dan vorige data, kunnen belangrijke parameters in een model geidentificeerd worden, waarbij toekomstige veranderingen een negatieve waarde mee krijgen. Het is echter onmogelijk, als we focussen op een ongelimiteerde sequentie aan taken, om alle informative te bewaren. Als een adaptieve methode voor specifieke testomgeving, stellen we voor om de belangrijkste parameters te leren tijdens het gebruik, terwijl het model actief is in de test omgeving. Het resultaat zal zijn dat het rampzalige 'alles' vergeten wordt vermeden, maar elegant selectief vergeten wordt

getolereerd. Voor een verdere ontwikkeling van toekomstige taken, bestuderen we de rol van spreiding in continu leren. We stellen een regulator voor die het percentage aan parameters voor elke taak significant beperkt en hierdoor ook het resultaat van het continu leren process aanzienlijk verbetert.

Aangezien de stapsgewijze toename van het leerproces niet in alle gevallen als uitgangspunt genomen kan worden bij continu leren, bestuderen we ook de setting van het on-line continu leren. We gaan uit van een oneindige stroom aan data, afkomstig van een continue distributie met een gecontroleerd of niet-gecontroleerd trainingssignaal. We stellen om te beginnen een protocol voor om ons werk van regularisatie van de belangrijkste parameters over te zetten op de on-line continu leren setting en daarmee laten we een verbetering in leren zien voor de verschillende stromen aan data.

Voor het geval van meer uitdagende situaties, waar de distributie van de input een grotere variëteit vertoont, onderzoeken we het gebruik van een vastgelegde buffer van voorbeelden uit de voorgaande reeksen. Wij stellen een voorbeeld van een selectiemethode voor, die geen veronderstellingen doet over de distributie van gegenereerde data. Voor zover we weten, waren we de eersten om het probleem van on-line continu leren aan te pakken.

De voorgestelde methoden in deze thesis hebben belangrijke aspecten van het continu leren aangepakt. Ze zijn geëvalueerd op verschillende benchmarks en voor verschillende leersequenties. Vooruitgang in de state-of-the-art van het continu leren werd gerealiseerd en de uitdaging om continu leren in de praktijk te kunnen toepassen werd onomstotelijk aangetoond.

List of Abbreviations

EBLL	Encoder Based Lifelong Learning
EG	Expert Gate
EWC	Elastic Weight Consolidation
LwF	Learning without Forgetting
GEM	Gradient Episodic Memory
iCaRL	Incremental Classifier and Representation Learning
i.i.d.	Independent and identically distributed
IMM	Incremental Moment Matching
MAP	Maximum A posteriori Probability estimate
MAS	Memory Aware Synapses
MLE	Maximum Likelihood Estimation
PCA	Principal Component Analysis
SCL	Selfless Continual Learning
SI	Synaptic Intelligence
SVD	Singular Value Decomposition

Contents

List of Figures

List of Tables

Chapter 1

Introduction

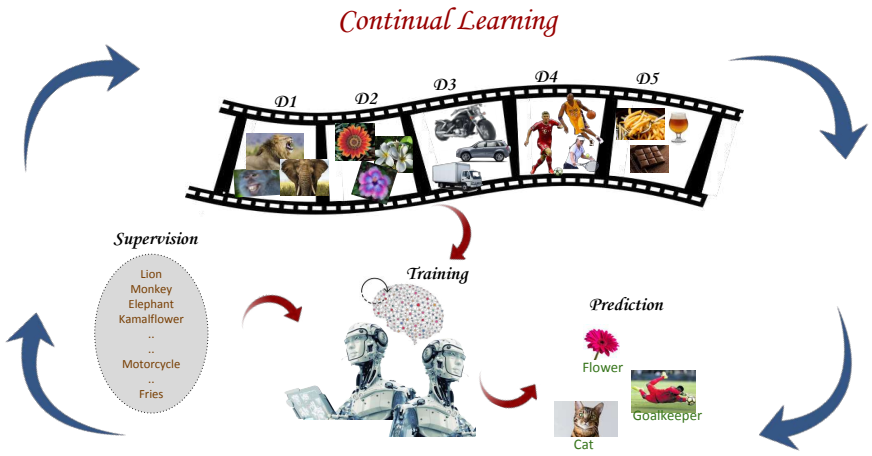


Figure 1.1: An illustration of the continual machine learning cycle. Data are received sequentially with optional supervision. The agent alters between learning and predicting. Red arrows represent the inner cycle that occurs whenever new data arrive.

Our world is complex, constantly changing and evolving and so are our brains, continually forming new organismic states to adapt and interact with this world. The evolution and self-organization depicted in living creatures resemble the essence of the difference between classical physics and physiology [51]. While classical physics studies describe a stationary world, physiology concerns with evolutionary systems and their non-stationary world. Our artificial agents have to be deployed and behave in this same dynamic non-stationary world. Without mechanisms allowing these agents

to constantly adapt and exploit new information, effective machine intelligence can't be realized.

Current machine learning models represented by neural networks are able to learn and even outperform human level performance in individual tasks, as in Atari games [141] and object recognition [133]. However, this learning process creates static neural models that are incapable of adapting and expanding their "function". Whenever new data are available, the training process of a neural network has to start all over again. In a world like ours, such a practice becomes intractable when moving to real scenarios where data are streaming, might be disappearing after a given period of time or even can't be stored at all due to storage constraints or privacy issues. Each day millions of images with new tags appear on social media. Every minute hundreds of hours of video are uploaded on Youtube. This new content contains new topics and trends that may be very different from what one has seen before - think e.g. of new emerging topics, fashion trends, social media hypes or technical evolution. This makes it crucial for neural networks to be able to adapt and be updated over time.

The main obstacle towards developing continually adapting systems is the "catastrophic forgetting" of old learned information once new knowledge is learned. Mccloskey and Ratcliff [101, 121] were the first to show catastrophic forgetting in neural networks, where the learning of new patterns of data results in a complete erase of the previously acquired knowledge. Catastrophic forgetting has also been attested in other machine learning models [52, 86]. However, the ability of neural networks to implicitly store the acquired knowledge in addition to its success and biological plausibility urge the need to study and understand the catastrophic forgetting phenomenon.

While natural cognitive systems can gradually forget old information, a complete loss of previous knowledge is rarely attested [42]. Humans tend to learn concepts sequentially one after another. Some concepts are revisited but this revisiting is unnecessary for a new concept to be conceived. Given current artificial neural networks, learning cannot occur in this sequential manner due to the catastrophic forgetting of previous concepts as new ones are learned. Typically, data of a given task are shuffled and performance largely increases with repeated revisiting over the training data. Since this shuffling and repeated re-visiting of all training data is clearly not the case for humans, that are able to learn and even exhibit better learning behavior when information is presented sequentially, French and Ferrara [43] posed the question of whether this immunity to catastrophic forgetting is present in other mammals. They showed that a learning of two time events sequentially in rats results in a complete wipe out of the first event once the second is learned. However, a concurrent learning of the same two events does show a forgetting but not the catastrophic forgetting shown in the sequential learning. This is very similar to what would occur if a simple neural network was used to model events presented sequentially. It has been suggested that overcoming of catastrophic forgetting in higher mammals could be due to the development of a hippocampal–neocortical separation [100, 43].

Catastrophic interference is a direct result of a more general problem in neural network, the so-called "stability–plasticity" dilemma [51]. While plasticity refers to the ability of integrating new knowledge, stability indicates the preservation of previous knowledge while new data is encoded. Hence, stability–plasticity is an essential building block in artificial and biological neural intelligent systems. *The main challenge is how to build intelligent systems that are dynamic and sensitive to new information while at the same time are stable and immune to catastrophic interference with previously acquired knowledge. Overcoming this challenge has been the driving goal of the work developed during the course of this PhD.*

1.1 Continual Learning

Continual learning, also referred to as lifelong learning, sequential learning or incremental learning, studies the problem of learning from an infinite stream of data stemmed from changing input domains and associated with different tasks, with the goal of using the acquired knowledge in problem solving and future learning [27]. The main criterion is the sequential nature of the learning process where only a small portion of input data from one or few tasks is available at once. It is impossible to label all training examples from all tasks before initiating the learning process and even if so, with a constantly evolving world, adaptation and continual learning is a must. For such a system or process to be efficient, all previously seen data should not be stored in their raw format and a full re-training at each point is simply infeasible at such a large scale.

Since the early development of neural networks, researchers studied the catastrophic forgetting problem and proposed that the parameters sharing which allows neural networks to generalize from seen data is the reason behind catastrophic forgetting [101, 121]. After learning one task, the network parameters correspond to one point in the parameter space. When learning a new task, the parameters will change their values to a new point that might not correspond to a solution to the first task. It has been shown that the parameter space of shallow networks contains cliffs in which small moves lead to a severe change in the function output [72].

Early research works developed several strategies to mitigate the forgetting under the condition of not storing the training data, mostly at a small scale of few examples and considering shallow networks [78, 12, 124]. Recently, after the revival of neural networks, the catastrophic forgetting problem and the continual learning paradigm received increased attention [85, 69, 134, 83]. We will first define the general continual learning setting and describe the main desired criteria of a continual learning system. We then move to point at the differences with other machine learning fields that share characteristics with continual learning.

General Continual Learning Setting. The general continual learning setting considers an infinite stream of data where at each time step t, the system receives a new sample(s) $\{x_t, y_t\}$ drawn non i.i.d. from a current distribution Q that could itself experience sudden or gradual changes.

The main goal is to learn a function f parameterized by θ that minimizes a predefined loss[1] ℓ on the new sample(s) without interfering with and possibly improving on those that were learned previously.

$$\theta^t = \underset{\theta,\xi}{\operatorname{argmin}}\, \ell(f(x_t;\theta), y_t) + \sum \xi_i \tag{1.1}$$

$$\text{s.t. } \ell(f(x_i;\theta), y_i) \leq \ell(f(x_i;\theta^{t-1}), y_i) + \xi_i, \tag{1.2}$$

$$\xi_i \geq 0 \quad ; \forall i \in [0\,..\,t-1]$$

Where $\xi = \{\xi_i\}$ is a slack variable that tolerates a small increase in some previous samples losses, those that are hard to maintain without affecting the learning of current samples .

1.1.1 Desiderata of Continual Learning

To build a machine learning system that achieves the goal of continual learning described in Equation 1.1, it is important to aim for some if not all the desired characteristics listed below. These characteristics facilitate the realization of a continual learning system.

1. **Constant memory.** The memory consumed by the continual learning paradigm should be constant w.r.t. the number of tasks or the length of the data stream. This avoids the need to deal with unbounded systems.

2. **No task boundaries.** Being able to learn from the input data without requiring a clear task division brings great flexibility to the continual learning method and makes it applicable to any scenario where data distribution is shifting and environment slowly changing.

3. **Online learning.** A largely ignored characteristic of continual learning is being able to learn from a continuous stream of data without offline training of large batches or separate tasks.

4. **Forward transfer.** This characteristic indicates the use of the previously acquired knowledge to aid the learning of new data/tasks.

[1] The loss itself might be learned or synthesized but this is left for future directions.

5. **Backward transfer.** A continual learning system shouldn't only aim at retaining previous knowledge but preferably improving the performance on previous tasks when learning future related tasks.

6. **Problem agnostic.** A continual learning method should be general and not limited to a specific setting (e.g. only classification).

7. **Adaptive.** Being able to learn from unlabeled data would increase the method applicability to cases where original training data no longer exist and open the door to a specific user setting adaptation.

8. **No test time oracle.** A well designed continual learning method shouldn't rely on a task oracle to perform prediction.

9. **Task revisiting.** When revisiting a previous task, the system should be able to successfully incorporate the new task knowledge.

10. **Graceful forgetting.** Given bounded system and infinite stream of data, a selective forgetting of unimportant information is needed to achieve a balance of stability and plasticity.

Due to the difficulty of the described continual learning problem and the various challenges that have to be dealt with, in order to meet the different desiderata, methods try to overcome "catastrophic forgetting" with different levels of relaxations. Of all desiderata, "online learning" is the most commonly violated due to the difficulty of strict per-example incremental learning. Therefore, a milder task incremental assumption is usually adopted.

Task Incremental Setting In this setting the data are streamed one task at a time, with different distributions for each task, while keeping the i.i.d. assumption and performing offline training within each task training phase. For each task, we are given a dataset $D_t = \{X^{(t)}, Y^{(t)}\}$ where $X^{(t)}, Y^{(t)} = \{x_n^{(t)}, y_n^{(t)}\}_{n=1}^{N_t}$ randomly drawn from a distribution Q_t of a current task T_t. The goal is to control the empirical risk of all seen tasks:

independent and identical distribution

$$R = \sum_{t=1}^{\mathcal{T}} \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(f(x_n^{(t)}; \theta), y_n^{(t)}) \tag{1.3}$$

where $\mathcal{T}$ is the number of tasks seen so far. Given a limited or no access to data from previous tasks, this can be expressed as minimizing the empirical risk of each new task

with the constraints of not increasing the loss of the previous tasks:

$$\begin{aligned}\theta^{\mathcal{T}} = \underset{\theta,\xi}{\operatorname{argmin}} \frac{1}{N_{\mathcal{T}}} \sum_{n=1}^{N_{\mathcal{T}}} \ell(f(x_n^{(\mathcal{T})};\theta), y_n^{(\mathcal{T})}) + \sum \xi_t \\ s.t. \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(f(x_n^{(t)};\theta), y_n^{(t)}) \leq \frac{1}{N_t} \sum_{n=1}^{N_t} \ell((f(x_n^{(t)};\theta^{\mathcal{T}-1}), y_n^{(t)}) + \xi_t, \\ \xi_t \geq 0 \quad ; \forall t \in [0\,..\,\mathcal{T}-1]\end{aligned} \tag{1.4}$$

Where $\xi = \{\xi_t\}$ is a slack variable that tolerates a small increase in some previous tasks losses. The word task here refers to an isolated training phase of a new batch of data that belongs to a new group of classes, a new domain or a different output space, e.g. scenes classification v.s. hand written digit classification. As such, following [60], a finer categorization can be used: incremental class learning where $P(Y^t) = P(Y^{t+1})$ but $\{Y^t\} \neq \{Y^{t+1}\}$ indicating disjoint labels in each task, incremental domain learning where $P(X^t) \neq P(X^{t+1})$ and $P(Y^t) = P(Y^{t+1})$ and the incremental task learning indicating $P(Y^t) \neq P(Y^{t+1})$ and $P(X^t) \neq P(X^{t+1})$.

1.2 Relation to Other Machine Learning Fields

The ideas of knowledge sharing, adaptation and transfer depicted in the outlined desiderata have been studied previously in machine learning and developed in isolated fields. We will describe each of them briefly and highlight the main differences with continual learning. See Figure 1.2 for an illustration of each related machine learning field setting.

Multi Task Learning. Multi-Task Learning considers the learning of multiple related tasks simultaneously using a set or a subset of shared parameters. It aims for a better generalization and less overfitting using the shared knowledge extracted from the related learned tasks. We refer to [170] for a survey on multi-task learning. Multi-task learning follows the offline training of all tasks with the presence of all tasks data at training time. It doesn't involve any adaptation after the multi-task model has been deployed, as opposed to continual learning.

Transfer Learning. Transfer learning aims at aiding the learning process of a given task by exploiting the knowledge of another task or domain. More formally, given a source domain with data distribution Q_S and its corresponding task T_S and a target

domain Q_T with task T_T, transfer learning aims to support the learning of the target task T_T in Q_T using the knowledge of Q_S and T_S, where $Q_S \neq Q_T$, or $T_S \neq T_T$. Transfer learning is mainly concerned with the forward transfer desiderata of continual learning. However, it doesn't involve any continuous adaptation after learning the target task. Moreover, the performance on the source task(s) is not taken into account during transfer learning. A quite popular example of transfer learning is finetuning, where models pre-trained on large tasks are used as initialization for tasks with limited training data.

Domain Adaptation. Domain adaptation is a sub-field of transfer learning where the source and target tasks are the same but drawn from different input domains. The target domain data is unlabelled and the goal is to adapt a model trained on the source domain to perform well on the unlabelled target domain. In other words, it relaxes the classical machine learning assumption of having training and test data drawn from the same distribution [30]. As mentioned above for transfer learning, domain adaptation is unidirectional and doesn't involve any accumulation of knowledge [27].

Learning to Learn (Meta Learning). The old definition of learning to learn was referring to the concept of improving the learning behavior of the model with training experience. According to [152], given a set of tasks, a model uses its experience from the past tasks to improve the performance on the current task. However, more recently, the common interpretation is the ability for a faster adaptation on a task with few examples given a large number of training tasks. While these ideas seem quite close to continual learning, meta learning follows the assumption of offline training (i.e. all tainting data are available at the same time) but with training data being tasks randomly drawn from a task training distribution and test data being test tasks with few examples. Hence, it is not capable, alone, of preventing forgetting on those previous tasks. An integration of the concept of meta learning with continual learning is an interesting area of research [66, 54].

Online Learning. Whereas in traditional offline learning, the entire training data has to be made available prior to learning the task, on the contrary online learning studies learning algorithms that learn to optimize predictive models over a stream of data instances sequentially. We refer to [137, 19] for surveys and overviews on the topic. In spite of considering a stream of data, online learning still assumes the i.i.d. data sampling procedure and just one task/domain, in contrast to continual learning.

Open World Learning. Open world learning [159, 16] deals with the problem of detecting new classes at test time, hence avoiding wrong assignments to known classes. When those new classes are then integrated in the model, it meets the problem of

continual learning. As such, open world learning can be seen as a sub task of continual learning.

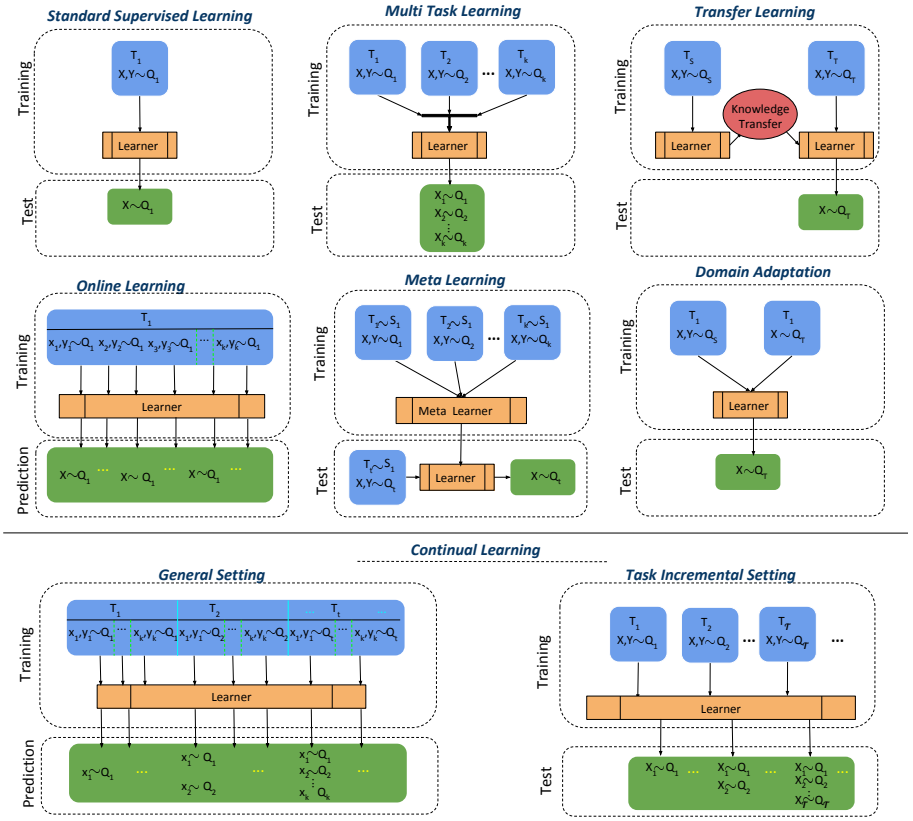


Figure 1.2: The main setup of each related machine learning field, illustrating the differences with general continual learning setting and task incremental setting.

1.3 Main Contributions

The goal of this thesis is to study the stability/plasticity dilemma in neural networks and to propose solutions to provide artificial agents with the ability to learn continually and accumulate the acquired knowledge over their lifetime. This is approached while aiming at achieving a reasonable trade-off between the characteristics described in Section 1.1.1.

We first consider the task incremental setting where tasks associated with their training and test data are received sequentially. Only access to the current task data is permitted, data from previous tasks are not retained. Under this setting, we start by proposing a system that allows for an expert level performance on each seen task while tolerating a linear increase in storage memory. A model is learned for each task and a gate is sequentially learned to assign the test samples to their corresponding models. This work was published as an article in the Computer Vision and Pattern Recognition conference, CVPR 2017. We then study the learning of the tasks sequence using a shared neural network model with fixed capacity. We propose a solution to learn a low dimensional manifold that captures the features needed for performing each task. Using the data of the new task as a proxy, features changes in the learned manifold are constrained while changes orthogonal to the manifold are free of charge. This work was presented as a conference paper at the International Conference on Computer Vision, ICCV 2017.

As to mitigate the effect of tasks being drawn from very different distributions and to avoid the need for a separate training phase to learn a lower dimensional feature space, we instead identify the important parameters in the neural network for a given task. The changes on the important parameters are penalized during the learning process. Important parameters can be estimated on any data opening the door for adaptive continual learning method. This work was published as a conference paper in the European Conference on Computer Vision, ECCV 2018. To account for future tasks when the capacity of the shared model is fixed, we study the role of sparsity in continual learning and propose a new regularizer to encourage the learning of a sparse representation. This work was published as a conference paper in the International Conference on Learning Representations, ICLR 2019.

We then move to study the more challenging online continual learning setting where data are streaming with no given task boundaries or offline training on big batches of data. We first propose a protocol that brings our important parameters regularizer to the online continual learning setting. The work was published in the Computer Vision and Patter Recognition conference, CVPR 2019. Next and as a solution to online continual learning when the data generating distribution is largely shifting, we rely on a buffer of historical samples to prevent forgetting. We present a method to select representative samples from the data stream using as a criterion, maximizing the samples diversity in the gradient space of the model parameters. This work published in the 2019 Conference on Neural Information Processing Systems, NeurIPS.

The rest of the manuscript is organized as follows: We present the background materials closely related to our contributions in Chapter 2. We then discuss from a broad perspective works on continual learning that emerged during the past few years in Chapter 3. Our work on sequential learning of experts is detailed in Chapter 4 while we discuss how to constrain the features space to reduce forgetting in Chapter 5. We move to present our method that estimates importance weights to the parameters of a neural network and penalizes their changes in Chapter 6. A study on the role of sparsity

and how to impose it in continual learning is described in Chapter 7. Under the online continual learning setting, we first present in Chapter 8 a protocol for deploying our importance weight based method. Then, our gradient based sample selection method is detailed in Chapter 9. We conclude and discuss the state of the art of continual learning, limitations and future directions in Chapter 10.

Chapter 2

Background

In this chapter we explain briefly background materials related to the works presented in this manuscript. We give a very short introduction to neural networks in Section 2.1 and to autoencoders in Section 2.2. We then remind the reader of parameters estimators and some popular regularizers in Section 2.3. We present a general Bayesian view on continual learning in Section 2.4. Knowledge distillation is explained in Section 2.5. We shortly define common continual learning terms in Section 2.6. Finally, we describe learning sequences and evaluation metrics used frequently in the experiments of this thesis (Section 2.7).

2.1 Neural Networks

Given a training dataset $D = \{x, y\}_{n=1}^{N}$ sampled i.i.d. from a target distribution $Q_{x,y}$, a neural network is employed to learn a function f that maps a given input x_n to a target output y_n. For example in handwritten digit recognition, x_n represents an image while y_n corresponds to the drawn digit. They are called networks because the learned function is composed of multiple functions, e.g. $f(x) = f_3(f_2(f_1(x)))$ where f_1 is the first layer, f_2 is the second layer and f_3 is the output layer. While the target y_n represents the desired output for the output layer, there is no target output for the other layers and they are called hidden layers. Each layer, represented by f_l, is parametrized by a weight vector $\theta_l = \{\theta_{ij}\}$ [1] with a vector input and a vector output, $f_l(x) = \phi(\theta_l x)$ where ϕ is the activation function. The layer can be viewed as a group of neurons, also called units or perceptrons, where each neuron maps a vector input to a scalar

[1] Note that we don't mention explicitly the bias, but it can be incorporated by padding the input to each layer by one.

output, hence the name multi layer perceptron, see Figure 2.1 for an example. ϕ is usually a non linear function except maybe from the output layer. By stacking multiple layers of neurons, one can approximate various non linear functions. In fact neural networks can be seen as universal approximators if provided with enough neurons in a hidden layer. The learning occurs through minimizing a loss function that represents the divergence between the output of the neural network and the target output. Due to the non-linearity of the neural networks, most cost functions become non-convex; and they are usually optimized by iterative gradient descent steps. The gradients of the loss function are computed through the backpropagation [131] of the estimated loss to the preceding layers of the network. Typically, the optimizer performs gradients steps on batches randomly sampled from the training set. This process is repeated and samples are re-visited until convergence. After a local minimum is reached, the neural network is deployed and used for performing predictions on new samples. We refer to [47] for a detailed introduction to neural networks.

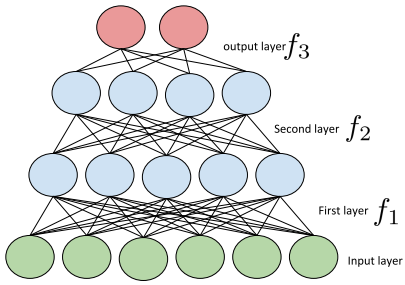


Figure 2.1: An example of a 3 layers neural network.

2.2 Autoencoders

An autoencoder [21] is a neural network that learns to produce an output similar to its input [47]. The network is composed of two parts: an encoder $h(x)$ which maps the input x to a code $c = h(x)$, and a decoder $g(c)$ that maps the code to a reconstruction of the input. The loss function $\ell(x, g(c))$ is simply the reconstruction error, e.g. the mean ℓ_2 distance between the inputs and their reconstructions. The encoder learns, through one or more hidden layers, a lower dimensonional representation (undercomplete autoencoder) or a higher dimensional representation (overcomplete autoencoder) of the input data. A

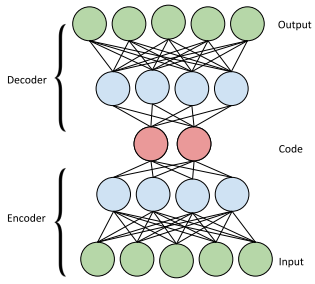


Figure 2.2: An undercomplete autoencoder with code size 2.

regularization term is necessary in overcomplete autoencders to prevent the autoencoders from copying their input. Figure 2.2 shows an example of undercomplete autoencoder. A linear autoencoder with a Euclidean loss function learns the same subspace as principal component analysis (PCA). However, autoencoders with non-linear functions yield better dimensionality reduction compared to PCA [58]. Autoencoders are usually used to learn feature representations in an unsupervised manner or for dimensionality reduction.

The low dimensional manifold learned by an undercomplete autoencoder represents mostly the variations that are needed to reconstruct relevant samples. It is maximally sensitive to variations in the data generating distribution but insensitive to changes orthogonal to the manifold. In Chapter 4 and Chapter 5, we exploit this characteristic of autoencoders to approach the continual learning problem.

2.3 Parameters Estimators & Popular Regularizers

Maximum likelihood Estimation

Given a dataset $D = \{x, y\}_{n=1}^{N}$ sampled i.i.d. from a target distribution $Q_{x,y}$ and a model with parameters $\theta = \{\theta_{ij}\}$, the likelihood of the data given the model parameters is defined as $L(\theta) = p(D|\theta)$.
A maximum likelihood estimator of the model parameters θ is:

$$\theta^{MLE*} = \max_{\theta} \quad p(D|\theta) \tag{2.1}$$

θ^{MLE*} is the model parameters that maximize the likelihood of the data D. Usually in optimization, we minimize an objective function, e.g. loss function. As such instead of maximizing the likelihood, we minimize the negative log likelihood.

$$\theta^{MLE*} = \min_{\theta} \quad -log(p(D|\theta)) \tag{2.2}$$

Regularization

The dataset used for training can be small and noisy [2] while the model is typically overparameterized. Minimizing the log-likelihood of the training data could result in overfitting, i.e. small training errors but higher test errors. To reduce overfitting, regularization is imposed to control the model variance without a significant increase in the bias. Many regularization techniques aim at limiting the model capacity by

[2] By noise we mean that the data points might not represent the true properties of the data generating distribution.

adding a penalty that corresponds to the norm of the model parameters. As examples we mention:

L2 regularization. An ℓ_2 norm penalty which is usually referred to as weight decay. It pulls the parameters towards the origin, keeping them with small magnitude. L2 regularization is also known as ridge regression or Tikhonov regularization. Minimizing the penalized objective would lead to:

$$\theta^{MLE*} = \min_{\theta} \quad -log(p(D|\theta)) + \frac{1}{2} \parallel \theta \parallel_2^2 \tag{2.3}$$

L1 regularization. Instead of ℓ_2 penalty, ℓ_1 norm is also a popular norm for regularization. It is defined as $\parallel \theta \parallel_1 = \sum_{i,j} |\theta_{ij}|$. ℓ_1 norm results in more sparse parameters than ℓ_2 norm and it has been used for feature selection as in the LASSO [153] (least absolute shrinkage and selection operator).

Note that usually a hyper parameter λ is used to weight the contribution of the regularizer. We explore the regularization effect on continual learning in Chapter 7.

Maximum a posteriori estimation

Bayes theorem estimates a distribution over the model parameters.

$$p(\theta|D) = \frac{p(\theta)p(D|\theta)}{p(D)} \tag{2.4}$$

where $p(\theta)$ is the prior that we want to impose when optimizing the parameters (θ) on D, $p(D|\theta)$ is the likelihood term, and $p(D)$ represents the marginal likelihood. For a given dataset, it is usually assumed that the marginal likelihood is constant.

$$p(\theta|D) \propto \quad p(\theta)p(D|\theta) \tag{2.5}$$

In a maximum a posteriori probability (MAP) estimate, we maximize the posterior:

$$\theta^{MAP*} = \max_{\theta} \quad p(\theta|D) \tag{2.6}$$

$$= \max_{\theta} \quad log\, p(\theta|D) \tag{2.7}$$

$$= \max_{\theta} \quad log\, p(D|\theta) + log\, p(\theta) \tag{2.8}$$

The prior imposed in the MAP represents our beliefs about the problem we want to solve.

Gaussian prior A popular choice is to assume a Gaussian prior. It presumes that each parameter θ_{ij} is an independent random variable with 0 mean and σ_{ij} standard deviation:

$$\theta_{ij} \sim \mathcal{N}(0, \sigma_{ij}^2) \tag{2.9}$$

$$p(\theta) = \prod_{i,j} p(\theta_{ij}) \tag{2.10}$$

$$log\left(p(\theta)\right) = \sum_{i,j} log(p(\theta_{ij})) \tag{2.11}$$

$$log\left(p(\theta)\right) = \sum_{i,j} log\frac{1}{\sqrt{2\pi\sigma_{ij}^2}} - \sum_{ij} \frac{1}{2\sigma_{ij}^2}|\theta_{ij}|^2 \tag{2.12}$$

$$log\left(p(\theta)\right) = C - \sum_{i,j} \frac{1}{2\sigma_{ij}^2}|\theta_{ij}|^2 \tag{2.13}$$

As such a MAP estimate with a Gaussian prior over the model parameters would be:

$$\theta^{MAP*} = \max_{\theta} \quad log\, p(D|\theta) + log\, p(\theta) \tag{2.14}$$

$$= \max_{\theta} \quad log\, p(D|\theta) - \sum_{i,j} \frac{1}{2\sigma_{ij}^2}|\theta_{ij}|^2 \tag{2.15}$$

If $\theta \sim \mathcal{N}(0, I)$ then the MAP estimator corresponds to the maximum likelihood estimator with L2 regularization.

$$\theta^{MAP*} = \max_{\theta} \quad log\, p(D|\theta) - \frac{1}{2} \parallel \theta \parallel_2^2 \tag{2.16}$$

$$= \min_{\theta} \quad - log\, p(D|\theta) + \frac{1}{2} \parallel \theta \parallel_2^2 \tag{2.17}$$

2.4 Continual Learning from a Bayesian Point of View

Suppose that we have a first dataset D_1 that corresponds to a given task T_1. According to Bayes rule, the posterior of the parameters is as follows:

$$log\, p(\theta|D_1) \propto \log\, p(D_1|\theta) + log\, p(\theta) \tag{2.18}$$

Suppose that after learning task T_1 and estimating its optimal parameters θ^1, e.g. using MAP estimator, we have received a new task T_2 with its dataset D_2 then:

$$log\, p(\theta|D_1, D_2) = \log\, p(D_2|\theta) + log\, p(\theta|D_1) - log(p(D_2|D_1)) \tag{2.19}$$

The first term is the log likelihood of the new task. Note that $log\, p(D_2|D_1)$ is a constant and can be ignored in the estimation of the parameters. Now the information

of the previous task is encoded in the posterior $p(\theta|D_1)$. The posterior represents the uncertainty of the parameters for that task. Estimating the true posterior is intractable. It can be approximated as a Gaussian distribution [110, 49]:

$$p(\theta|D_1) = \mathcal{N}(\theta^1, \Sigma) \tag{2.20}$$

with the mean equal to the optimal parameters for the previous task and a diagonal covariance matrix assuming independent parameters.

The objective function of the new task is then:

$$\max_{\theta} \quad \log p(D_2|\theta) - \frac{1}{2}(\theta - \theta^1)^\intercal \Sigma^{-1}(\theta - \theta^1) \tag{2.21}$$

$$\max_{\theta} \quad \log p(D_2|\theta) - \sum_{i,j} \frac{1}{2\sigma_{ij}^2}(\theta_{ij} - \theta_{ij}^1)^2 \tag{2.22}$$

While σ_{ij}^2 corresponds to the uncertainty of the parameter θ_{ij}, $\frac{1}{\sigma_{ij}^2}$ represents the parameter's importance for the previous task. Assuming an equal parameters importance, Σ can be set to the identity matrix I. As such, optimizing the previous objective would encourage all the parameters to stay close to their optimal values at the previous task θ^1. However, it has been shown to be vulnerable to catastrophic forgetting [69, 49]. As proposed by [69], Laplace approximation can be made assuming the diagonal precision matrix equal to the diagonal Fisher information matrix F where $\frac{1}{\sigma_{ij}^2} = F_{ij}$. Other methods have proposed different ways of estimating Σ^{-1} or, as we shall call it the importance weights Ω. The methods that assume a Gaussian prior to prevent forgetting are called prior focused methods as we show in the next chapter.

2.4.1 Fisher Information Matrix

The Fisher information matrix represents the covariance of the gradient of the model's log likelihood function with respect to points sampled from the model's distribution [99]. Given a dataset D sampled i.i.d. from a target distribution $Q_{x,y}$, we want to learn a model with distribution $P_{x,y}(\theta)$ that maps the observations $\{x_n\}$ to their corresponding labels $\{y_n\}$. The Fisher information matrix of $P_{x,y}(\theta)$ is then:

$$\mathcal{F} = E_{P_{xy}}\Big[\nabla \log\, p(x, y|\theta) \nabla \log\, p(x, y|\theta)^\intercal\Big] \tag{2.23}$$

Estimating the Fisher information matrix requires sampling from the model's distribution. To avoid extra computation cost, an approximation of the Fisher information matrix called empirical Fisher information is usually used.

Empirical Fisher information

The empirical Fisher employs cases sampled from the dataset instead of the model's distribution. As such, the expectation is computed over the target distribution Q_{xy} instead of the model's distribution $P_{xy}(\theta)$ [99].

$$\bar{\mathcal{F}} = E_{Q_{xy}}\Big[\nabla \log\, p(x,y|\theta)\nabla \log\, p(x,y|\theta)^{\intercal}\Big] \tag{2.24}$$

$$= E_{Qx}\Big[E_{Q_y}\Big[\nabla \log\, p(y|x,\theta)\nabla \log\, p(y|x,\theta)^{\intercal}\Big]\Big] \tag{2.25}$$

$$= \frac{1}{N}\sum_{n}\Big[\nabla \log\, p(y_n|x_n,\theta)\nabla \log\, p(y_n|x_n,\theta)^{\intercal}\Big] \tag{2.26}$$

In [69] the inverse of the diagonal of the Fisher information or its empirical approximation is used to estimate the covariance of the prior distribution, i.e. the inverse of the previous task importance weights. In Chapter 6, we propose an alternative of estimating the importance weights.

2.5 Knowledge Distillation

Knowledge distillation [57] is a technique that aims at transferring the knowledge embedded in one model to another. Usually, a big model or an ensemble of models is used to learn a function from a big training dataset, which is quite computationally demanding. This technique aims at distilling the knowledge from the big/ensemble to a smaller model that is much lighter to deploy. The probabilities of each class in the output layer of the big model convey richer information than the label of the image. It also captures the similarities/dissimilarities between the different classes that are learned by the large model. Suppose that $\hat{y}$ is the output of the light model and y^* is the output of the large model. Then the knowledge distillation loss is defined as:

$$\ell_{dist}(\hat{y}, y^*) = -\langle z^*, \log \hat{z}\rangle \tag{2.27}$$

where $\log$ is operated entry-wise and

$$z_i^* = \frac{y_i^{*1/\tau}}{\sum_j y_j^{*1/\tau}} \text{ and } \hat{z}_i = \frac{\hat{y}_i^{1/\tau}}{\sum_j \hat{y}_j^{1/\tau}} \tag{2.28}$$

where $i, j \in \{1, .., J\}$ running over the output layer units, τ is the temperature. The application of a high temperature τ softens the probability distribution over the classes. Knowledge distillation can be used to reduce forgetting by distilling the knowledge from a previous model to a model being trained on a new task, as we show in Chapter 5.

2.6 Continual Learning Terminology

In this manuscript, we use terms that have become common in the continual learning community. Table 2.1 gives a short definition of each frequently used terms.

Term	Description
`Problem`	Describes what a machine learning model is employed to solve, e.g. the problem of image classification.
`Task`	A more specific term than "problem" with a predefined input and output. For example, scene classification is a task of the image classification problem.
`Domain`	Indicates a unique data generating distribution of a given task input.
`Environment`	A physical world in which a machine learning agent operates; it is mostly used in reinforcement learning.
`Joint Training`	Indicates the training of a shared model on multiple tasks using all their data.
`Fine-tuning`	Indicates training a given task using as initialization a model pre-trained on another task, typically with large amount of labelled data. It is the most typical form of transfer learning.
`Catastrophic Forgetting`	The complete erase of the previously acquired knowledge once new knowledge is learned [42].
`Catastrophic Interference`	A similar terminology to catastrophic forgetting, indicating that the new learning process has destructively affected the previous learning and caused catastrophic forgetting.
`Replay`	When learning a new task or new samples, samples from previous history are re-intorduced to the learner.
`Rehearsal`	Performing learning steps on the replayed samples from previous history when learning a new task or new samples.
`Pseudo Rehearsal`	Performing learning steps on generated samples, mimicking previous history, when learning a new task or new samples.
`Revisiting`	When a learner encounters samples from previous tasks or previous distributions again. This doesn't indicate that the exact same samples are reintroduced.

Table 2.1: Terminology: list of the main terms used in this manuscript with a brief description each.

2.7 Continual Learning Evaluation

Datasets We provide a short description of datasets repeatedly used in the experiments of this thesis and in the continual learning literature. See Figure 2.3 for example images.

1. *ImageNet* (LSVRC 2012 subset) [133], is an image database organized according to the WordNet hierarchy which has more than 1 million training images and 1000 classes of different objects. It is a big dataset. Models usually are pre-trained on ImageNet and then finetuned on other tasks.

2. MIT *Scenes* [117] for indoor scene classification. It is composed of 67 Indoor categories with 100 images per category divided into 80 images for training and 20 images for test.

3. Caltech-UCSD *Birds* [157] for fine-grained bird classification with 200 bird species (mostly North American) and a total number of 5,994 samples.

4. Oxford *Flowers* [111] for fine-grained flower classification. It is composed of 102 different categories of flowers with 2040 training samples and more than 6000 test samples.

5. Stanford *Cars* dataset [73] for fine-grained car classification. It contains 16185 images of 196 classes of cars split into a half for training and half for testing.

6. FGVC-*Aircraft* dataset [94] for fine-grained classification of aircraft, this dataset contains 10200 images of aircraft belonging to about 100 different model variants.

7. VOC *Actions*, the human action classification subset of VOC challenge 2012 [36]. It has over 6000 train and test images. It is composed of 10 action classes and one bin class for other non classified actions. This dataset has multi-label annotations. For sake of consistency, we only use the actions with single label.

8. *SVHN* [108] the Street View House Numbers, a dataset for digit recognition. SVHN is obtained from house numbers in Google Street View images. It has over 600,000 digit images. Each is a 32-by-32 image centered around a single character.

9. *Letters* [31] the Chars74K dataset for character recognition in natural images. We use the English set and exclude the digits, considering only the characters. It has 52 classes both lower and upper case letters, and over 10000 training images with 1151 test samples.

10. *MNIST* dataset [82] for handwritten digit classification. It has a training set of 60,000 examples, and a test set of 10,000 examples. Images are 28-by-28 with black background and white digits.
11. *Cifar-10* dataset [75] consists of 60,000 32-by-32 colour images in 10 classes of different objects, with 6000 images per class. There are 50,000 training images and 10,000 test images.
12. *Cifar-100* dataset [75] is similar to Cifar-10 but with 100 classes grouped into 20 super categories. Each class has 600 images split into 500 for training and 100 for test.

For *Cars*, *Aircraft* and *Actions*, we use the bounding boxes instead of the full images as the images might contain more than one object.

Benchmarks Continual learning aims at developing methods that are able to accumulate knowledge over time. To benchmark a continual learning method, datasets mentioned above are usually grouped or split to form a sequence of tasks. There are three main typical ways of creating tasks sequences:

- Pixel perumutation: Given a dataset, a first task is formed from this dataset T_1. Then, a sequence of T tasks is created with the same dataset images but with unique pixel permutations each. The goal is to perform the task on all the different permutations that are learned sequentially. One of the most popular examples is permuted MNIST based on MNIST dataset.
- Group of datasets: By arranging a group of different datasets, a sequence of tasks is formed. The goal is to learn the different tasks sequentially and be able to perform all the tasks. One of the frequently used sequences is an 8 tasks sequence, we proposed in Chapter 6. The sequence is as follows: Flowers→Scenes→Birds→Cars→Aircraft→Actions→Letters→SVHN.
- A split of one dataset: Another scenario is to split one dataset into multiple groups of classes forming a sequence of tasks. For example, splitting Cifar-100 into 10 tasks where each task is composed of 10 classes.

Evaluation metrics Continual learning is an emerging filed; recently multiple evaluation metrics have been proposed to measure different aspects of continual learning. However, throughout this manuscript, we focus on two simple, yet important measures:

- Accuracy: At the end of a learned sequence, the accuracy of each learned task is computed in addition to the the average accuracy on all the tasks.

- Forgetting: The difference between the accuracy of a given task once learned and its accuracy at the end of the learned sequence. An average forgetting can also be reported. The performance of a previous task could be better than its performance once learned and a negative forgetting is then measured, usually referred to as backward transfer. However, this is rarely the case since a degradation in performance is typically attested.

Figure 2.3: Sample images from datasets used in this manuscript.

Chapter 3

Related Work

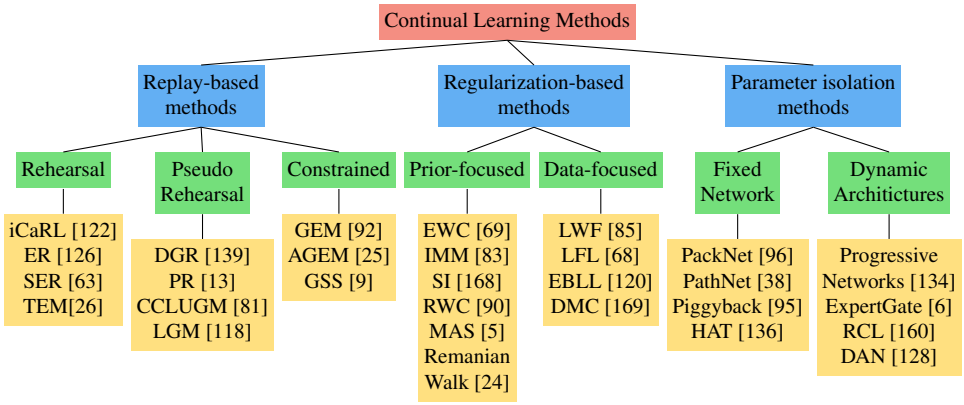
As we explained in the introduction, Chapter 1, continual learning studies the problem of learning from an infinite stream of data drawn non i.i.d. from a distribution Q_t that itself experiences sudden or gradual changes. These changes correspond to different input domains, classes or tasks switching from T_t to T_{t+1}.

Early works approaching continual learning focused on the catastrophic interference problem when learning sequentially examples of different input patterns (e.g. of different categories). Several directions have been explored such as reducing representation overlap [39, 40, 78, 79, 144], replay samples or virtual samples from previous history [124, 142] or dual architectures [41, 130, 12]. However, these works mainly considered few examples (in the order of tens) based on specific shallow architectures.

After the success of deep learning thanks to the powerful GPUs allowing training on large datasets and deep architectures, continual learning and catastrophic forgetting have received increased attention. [49] studied empirically the forgetting when learning two tasks sequentially using different activation functions and dropout regularization [146]. It was concluded that dropout mitigates forgetting with possible advantages to activation functions that encourage neurons competition such as Maxout [50] and LWTA [147]. [112] studied incremental task learning from a theoretical perspective with the goal of transferring knowledge to future tasks.

More recent works have addressed continual learning with longer sequences of tasks and large number of examples. The various developed methods can be categorized into three major families based on how the information of previous data is stored and used in future learning and prediction. A first family are the *replay* based methods. They store the information in the example space either directly in a replay buffer, or compressed

Figure 3.1: A tree diagram illustrating the different continual learning families of methods and the different branches in each family. Leaves list example methods.



in a generative model. When learning from new data, old examples are reproduced from the replay buffer or the generative model, and used for rehearsal/retraining or to constrain the current learning step. We call this category replay instead of rehearsal since conventionally rehearsal implies retraining, yet the old examples can also be used to provide constraints [92]. A second family are the *regularization* based methods. They consolidate the knowledge stored in the learned model through imposing a functional or parameter regularization term. We term the last major family as *parameter isolation*. They use isolated parameters for different tasks to prevent interference. Dynamic architectures that freeze/grow the network belong to this family. Below, we describe generally the core concepts and the main methods of each family while leaving to each chapter the detailed description of its closely related works. Figure 3.1 illustrates the different families of continual learning methods and their sub-groups with example methods each.

3.1 Replay-based Methods

As the name indicates the replay based methods replay memories from previous history during the learning from new data. Although storage of the original examples in memory for rehearsal dates back to the 1990s [124], to date it is still a rule of thumb to overcome catastrophic forgetting in practical problems. For example, experience replay is widely used in reinforcement learning where the training distribution is usually non-stationary and learning is apt to forgetting [87, 105].

In the context of incremental classification, iCaRL[122] maintains a subset of exemplars

per class that are rehearsed when new classes are learned. To ensure a bounded system, the number of stored exemplars is set to a fixed budget through a feature mean matching selection criterion. In settings where data is streaming with no clear task boundaries, [126] suggests the use of reservoir sampling to limit the number of stored samples to a fixed budget assuming an overall i.i.d. distributed data stream.

While rehearsal might be prone to overfitting and seems to be bounded by joint training when all previous data is available, constrained optimization is an alternative solution that leaves more room for forward/backward transfer. As proposed in GEM [92] under the task incremental setting, the key idea is to constrain the parameters update on the new task to not interfere with the previous tasks. This is achieved through projecting the estimated gradient direction in the feasible region outlined by previous tasks gradients through a first order Taylor series approximation. AGEM [25] has relaxed the problem to projection on one direction estimated by randomly selected samples from a buffer of previous tasks data. We have recently extended this solution to a pure online continual learning setting where no task boundaries are known and have proposed to select a subset of samples that maximally approximate the feasible region of the historical data, as we will show in Chapter 9.

In the absence of previous samples, pseudo rehearsal is an alternative strategy used in the early works with shallow neural networks. Random inputs and the outputs of previous model(s) given these inputs are used to approximate the previous tasks samples [124]. With deep networks and large input vectors (e.g. full resolution images) random input cannot cover the input space [13]. Recently, generative models have shown the ability to generate high quality images [48, 18] which opened up the possibility to model the data generating distribution and retrain on the generated examples [139]. However, this also adds to the complexity of training the generative model continually, with extra care needed to balance the retrieved examples and avoid the mode collapse problem.

3.2 Regularization-based Methods

When no storage of raw input is possible, an important line of works proposes an extra regularization term to consolidate the previous knowledge when learning from new data. The constraint of not storing any historical sample is mainly motivated by privacy reasons, as in the case of medical applications, in addition to being memory efficient. We can further divide these methods into data focused and prior focused methods.

Data-focused Methods

The basic building block in data-focused methods is the knowledge distillation from a previous model (trained on a previous task) to the model being trained on the new data. It was first proposed by [142] to use the output of previous tasks models given new task input images mainly for improving the new task performance. It has been re-introduced in LwF [85] to mitigate the forgetting and for knowledge transfer. This is done by using the outputs of the previous model as soft labels for previous tasks. Other works [68, 169] have been introduced with related ideas, however, as we will show in Chapter 4; this strategy might be vulnerable to domain shift between tasks. Therefore, we have suggested an additional term to constrain the features of each task in their own learned low dimensional manifold (see Chapter 5).

Prior-focused methods

To mitigate the forgetting, prior focused methods estimate a distribution over the model parameters which is used as the prior distribution when learning from new data, following the Bayesian view on continual learning described in Chapter 2, Section 2.4. As this quickly becomes infeasible w.r.t. the number of parameters in deep neural networks, parameters are usually assumed independent and an importance weight is estimated for each parameter in the neural network. During the training of later tasks, changes to important parameters are penalized. Elastic weight consolidation (EWC) [69] was the first to establish this approach. Variational Continual Learning (VCL) has introduced a variational framework for this family [110]. Synaptic Intelligence (SI) [168] estimates the parameters importance during the training of a task based on the contribution of their update to the decrease of the loss. In Chapter 6 we explain our approach to estimate online the importance weights based on unlabelled data, which allows for user adaptive settings and adds great flexibility in deploying the method. While the prior focused family relies on tasks boundaries to estimate the prior distribution, we further extend our work to task free settings, as we shall explain in Chapter 8.

Overall, the soft penalty introduced in the regularization family might not be sufficient to restrict the optimization process to stay in the feasible region of the previous tasks, especially with long sequences [37], which might result in an increased forgetting of earlier tasks.

3.3 Parameter Isolation Methods

To prevent any possible forgetting of the previous tasks, in parameter isolation methods, different subsets of the model parameters are dedicated to each task. When there is no constraints on the size of the architecture, this can be done by freezing the set of parameters learned after each previous task and growing new branches for new tasks [134, 160]. Our Expert Gate [6] is a representative work that ensures no forgetting by design and aims at optimal performance for future tasks through knowledge transfer, as we will show in Chapter 4.

Alternatively, under a fixed architecture, methods proceed by identifying the parts that are used for the previous tasks and masking them out during the training of the new task. This can be either imposed at the parameters level [96, 38] or at the neurons level as proposed in [136]. These methods usually require a task oracle to activate the corresponding masks or task branch at prediction time. Our Expert Gate [6] avoids this problem through learning an auto-encoder gate. In general, this family is restricted to the task incremental setting and better suited for learning long sequences of tasks when models capacity is not constrained and optimal performance is a priority.

Chapter 4

Sequentially Learning a Network of Experts

In this chapter, we consider the task incremental setting and set the target of achieving an optimal performance on each seen task. We argue that specialized models are needed for expert level performance and develop a gate, learned sequentially, to forward the test samples to the corresponding specialized model at test time. We evaluate our system on image classification and video prediction problems, and demonstrate the desired expert level performance on all the learned tasks. This work was published as an article in CVPR 2017 [6].

4.1 Introduction

In the age of deep learning and big data, we face a situation where we train ever more complicated models with ever increasing amounts of data. We have different models for different tasks trained on different datasets, each of which is an expert on its own domain, but not on others. In a typical setting, each new task comes with its own dataset. Learning a new task, say scene classification based on a pre-existing object recognition network trained on ImageNet, requires adapting the model to the new set of classes and fine-tuning it with the new data. The newly trained network performs well on the new task, but has a degraded performance on the old ones due to the catastrophic forgetting problem.

Ideally, a system should be able to operate on different tasks and domains and give the best performance on each of them. For example, an image classification system that

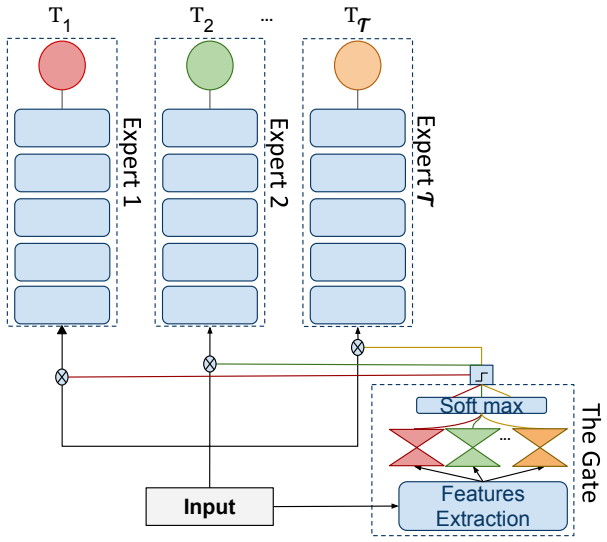


Figure 4.1: The architecture of our Expert Gate system.

is able to operate on generic as well as fine-grained classes, and in addition performs action and scene classification. If all previous training data were available, a direct solution would be to jointly train a model on all the different tasks or domains. Each time a new task arrives along with its own training data, new layers/neurons are added, if needed, and the model is retrained on all the tasks. Such a solution has three main drawbacks. The first is the risk of the negative inductive bias when the tasks are not related or simply adversarial. Second, a shared model might fail to capture specialist information for particular tasks as joint training will encourage a hidden representation beneficial for all tasks. Third, each time a new task is to be learned, the whole network needs to be re-trained. Apart from the above drawbacks, the biggest constraint with joint training is that of keeping all the data from the previous tasks. This is a difficult requirement to be met for a continual learning system, especially in the era of big data.

For example, ILSVRC [132] has 1000 classes, with over a million images, amounting to 200 GB of data. Yet the AlexNet model trained on the same dataset, is only 200 MB, a difference in size of three orders of magnitude. With increasing amounts of data collected, it becomes less and less feasible to store all the training data, and more practical to just store the models learned from the data.

Without storing the data, one can consider regularization strategies for continual learning. While this can mitigate the forgetting of old tasks, it is unlikely to maintain the achieved performance when learning scales to long sequences causing a bias

towards the new tasks and a possible buildup of errors on the older ones, as we show in our experiments, Section 4.4. Moreover, regularization strategies suffer from the same drawbacks as the joint training described above. Instead of having a network that is jack of all trades and master of none, to obtain the best performance for each task we propose having different specialist or expert models for different tasks, as also advocated in [57, 65, 134]. Therefore we build a Network of Experts, where a new expert model is added whenever a new task arrives and knowledge is transferred from previous models.

With an increasing number of task specializations, the number of expert models increases. Modern GPUs, used to speed up training and testing of neural nets, have limited memory (compared to CPUs), and can only load a relatively small number of models at a time. We obviate the need for loading all the models by learning a gating mechanism that uses the test sample to decide which expert to activate (see Figure 4.1). For this reason, we call our method *Expert Gate*.

Unlike [71], who train one Uber network for performing vision tasks as diverse as semantic segmentation, object detection and human body part detection, our work focuses on tasks with a similar objective. For example, imagine a drone trained to fly through an environment using its frontal camera. For optimal performance, it needs to deploy different models for different environments such as indoor, outdoor or forest. Our gating mechanism then selects a model on the fly based on the input video. Another application could be a visual question answering system, that has multiple models trained using images from different domains. Here too, our gating mechanism could use the data itself to select the associated task model.

Even if we could deploy all the models simultaneously, selecting the right expert model is not straightforward. Just using the output of the highest scoring expert is no guarantee for success as neural networks can erroneously give high confidence scores, as shown in [109]. We also demonstrate this in our experiments (Section 4.4). Training a discriminative classifier to distinguish between tasks is also not an option since that would again require storing all training data. What we need is a task recognizer that can tell the relevance of its associated task model for a given test sample. This is exactly what our gating mechanism provides. In fact, also the prefrontal cortex of the primate brain is considered to have neural representations of task context that act as a gating in different brain functions [97].

We propose to implement such task recognizer using an undercomplete autoencoder as a gating mechanism. We learn for each new task or domain, a gating function that captures the shared characteristics among the training samples and can recognize similar samples at test time. We do so using a one layer undercomplete autoencoder. Each autoencoder is trained along with the corresponding expert model and maps the training data to its own lower dimensional subspace. At test time, each task autoencoder projects the sample to its learned subspace and measures the reconstruction error due

to the projection. The autoencoder with the lowest reconstruction error is used like a switch, selecting the corresponding expert model (see Figure 4.1).

Interestingly, such autoencoders can also be used to evaluate *task relatedness* at training time, which in turn can be used to determine which prior model is more relevant to a new task. We show how, based on this information, Expert Gate can decide which specialist model to transfer knowledge from when learning a new task and whether to only use fine-tuning or a dedicated regularization strategy [85].

To summarize, the contributions of this chapter are the following. We develop Expert Gate, a continual learning system that can sequentially deal with new tasks without storing all previous data. It automatically selects the most related prior task to aid learning the new task. At test time, the appropriate model is loaded automatically to deal with the task at hand. We evaluate our gating network on image classification and video prediction problems.

The rest of the chapter is organized as follows. We discuss related work in Section 4.2. Expert Gate is detailed in Section 4.3, followed by experiments in Section 4.4. We finish with concluding remarks and a summary of the chapter in Section 4.5.

4.2 Related Work

Our end goal is to develop a system that can reach expert level performance on multiple tasks, with tasks learned sequentially. As such, it lies at the intersection between multi-task learning and incremental task learning.

Multi-task learning. In multi-task learning often one shared model is used for all tasks. This has the benefit of relaxing the number of required samples per task but could lead to suboptimal performance on the individual tasks. On the other hand, multiple models can be learned, that are each optimal for their own task, but utilize inductive bias / knowledge from other models [22].

To determine which related tasks to utilize, [151] clusters the tasks based on the mutual information gain when using the information from one task while learning another. This is an exhaustive process. As an alternative, [64, 162, 80] assume that the parameters of related tasks models lie close by in the original space or in a lower dimensional subspace and thus cluster the tasks parameters. They first learn tasks models independently, then use the tasks within the same cluster to help improving or relearning their models. This requires learning individual tasks models first. Alternatively, we use our tasks autoencoders, that are fast to train, to identify related tasks.

Multiple models for multiple tasks. One of the first examples of using multiple models, each one handling a subset of tasks, was by Jacobs et al. [65]. They trained an adaptive mixture of experts (each a neural network) for multi-speaker vowel recognition and used a separate gating network to determine which network to use for each sample. They showed that this setup outperformed a single shared model. A downside, however, was that each training sample needed to pass through each expert, for the gating function to be learned. To avoid this issue, a mixture of one generalist model and many specialist models has been proposed [3, 57]. At test time, the generalist model acts as a gate, forwarding the sample to the correct network. However, unlike our model, these approaches require all the data to be available for learning the generalist model, which needs to be retrained each time a new task arrives.

Incremental task learning. all previous knowledge is used, regardless of task relatedness.

Our system is immune to forgetting by design since each task has its own specialist model and we obviate the need for storing all the training data collected during the lifetime of an agent, by learning task autoencoders that learn the distribution of the task data, and hence, also capture the meta-knowledge of the task.

Our expert gate belongs to the parameter isolation family (see Chapter 3, Section 3.3). In this family, two architectures, namely the progressive network [134] and the modular block network [150], also use multiple networks, a new one for each new task. They add new networks as additional columns with lateral connections to the previous nets. These lateral connections mean that each layer in the new network is connected to not only its previous layer in the same column, but also to previous layers from all previous columns. This allows the networks to transfer knowledge from older to newer tasks. However, in these works, choosing which column to use for a particular task at test time is done manually, and the authors leave its automation as future work. Here, we propose to use an autoencoder to determine which model, and consequently column, is to be selected for a particular test sample.

4.3 The Proposed Method

We consider the case of incremental task learning where tasks and their corresponding training data arrive sequentially. For each task, we learn a specialized model (expert) by transferring knowledge from previous tasks – in particular, we build on the *most related* previous task. Simultaneously we learn a gating function that captures the characteristics of each task. This gate forwards the test data to the corresponding expert resulting in a high performance over all learned tasks.

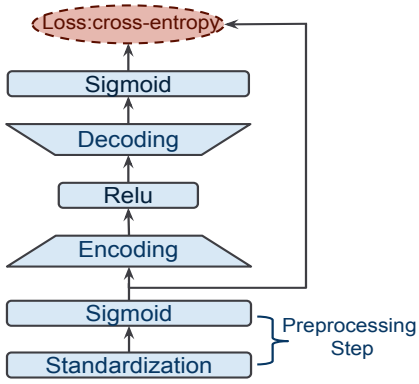


Figure 4.2: The deployed autoencoder gate structure.

The main question then is: how to learn such a gate function to differentiate between tasks, without having access to the training data of previous tasks? To this end, we learn a low dimensional subspace for each task/domain. At test time we then select the representation (subspace) that best fits the test sample. We do that using an undercomplete autoencoder per task. Below, we first describe this autoencoder in more detail (Section 4.3.1). Next, we explain how to use them for selecting the most relevant expert (Section 4.3.2) and for estimating task relatedness (Section 4.3.3).

4.3.1 The Autoencoder Gate

As we explained in Chapter 2 Section 2.2, an autoencoder [21] is a neural network that learns to produce an output similar to its input [47].

Autoencoders are usually used to learn feature representations in an unsupervised manner or for dimensionality reduction. Here, we use them for a different goal. The lower dimensional manifold learned by one of our undercomplete autoencoders will be maximally sensitive to variations observed in the task data but insensitive to changes orthogonal to the manifold. In other words, it represents only the variations that are needed to reconstruct relevant samples. Our main hypothesis is that the autoencoder of one domain/task should thus be better at reconstructing the data of that task than the other autoencoders. Comparing the reconstruction errors of the different tasks autoencoders then allows to successfully forward a test sample to the most relevant expert network. It has been stated by [4] that in regularized (over-complete) autoencoders, the opposing forces between the risk and the regularization term result in a score like behavior for the reconstruction error. As a result, a zero reconstruction

loss means a zero derivative which could be a local minimum or a local maximum. However, we use an unregularized one-layer undercomplete autoencoder and for these, it has been shown [98] that the mean squared error criterion we use as reconstruction error estimates an energy function that outputs low values for input samples similar to training data, and high energy values otherwise. In fact, there is no need in such a one-layer autoencoder to add a regularization term to pull up the energy on unseen data because the narrowness of the code already acts as an implicit regularizer that prevents the autoencoder from learning the identity function.

Preprocessing We start from a robust image representation $F(x)$, namely the activations of the last convolutional layer of AlexNet pretrained on ImageNet. Note that other feature extraction methods can be used. Before the encoding layer, we pass this input through a preprocessing step, where the input data is standardized, followed by a sigmoid function. The standardization of the data, i.e. subtracting the mean and dividing the result by the standard deviation, is essential as it increases the robustness of the hidden representation to input variations. Normally, standardization is done using the statistics of the data that a network is trained on, but in this case, this is not a good strategy since at test time, we compare the relative reconstruction errors of the different autoencoders. Different standardization regimes lead to non-comparable reconstruction errors. Instead, we use the statistics of ImageNet for the standardization of each autoencoder. Since this is a large dataset, it gives a good approximation of the distribution of natural images. After standardization, we apply the sigmoid function to map the input to a range of $[0\,1]$.

Network architecture We design a simple autoencoder that is no more complex than one layer in a deep model, with a one layer encoder/decoder (see Figure 4.2). The encoding step consists of one fully connected layer followed by ReLU [167]. We make use of ReLU activation units as they are fast and easy to optimize. ReLU also introduces sparsity in the hidden units which leads to better generalization. For decoding, we use again one fully connected layer, but now followed by a sigmoid. The sigmoid yields values between $[0\,1]$, which allows us to use binary cross entropy as the loss function. At test time, we use the Euclidean distance to compute the reconstruction error.

4.3.2 Selecting the Most Relevant Expert

At test time, and after learning the autoencoders $\{A_t\}_{t=1}^{\mathcal{T}}$ for the seen tasks, we add a softmax layer that takes as input the reconstruction errors $\{er_t\}$ from the different tasks autoencoders $\{A_t\}$ given a test sample x. The reconstruction error er_t of the t-th autoencoder is the Euclidean distance between output of the autoencoder and the representation of the input sample $F(x)$. The softmax layer gives a probability for each

task autoencoder indicating its confidence:

$$p_a = \frac{e^{(-er_a/\tau)}}{\sum_t e^{(-er_t/\tau)}} \tag{4.1}$$

where τ is the temperature. We use a temperature value of 2 as in [57, 85] leading to soft probability values. Given these confidence values, we load the expert model associated with the most confident autoencoder. For tasks that have some overlap, it may be convenient to activate more than one expert model instead of taking the max score only. This can be done by setting a threshold on the confidence values, see section 4.4.2.

4.3.3 Measuring Task Relatedness

Given a new task $T_{\mathcal{T}}$ associated with its data $D_{\mathcal{T}}$, we first learn an autoencoder for this task $A_{\mathcal{T}}$. Let T_t be a previous task with associated autoencoder A_t. We want to measure the task relatedness between task $T_{\mathcal{T}}$ and task T_t. Since we do not have access to the data of task T_t, we use the validation data from the current task $T_{\mathcal{T}}$. We compute the average reconstruction error $Er_{\mathcal{T}}$ on the current task data made by the current task autoencoder $A_{\mathcal{T}}$ and, likewise, the average reconstruction error Er_t made by the previous task autoencoder A_t on the current task data. The relatedness between the two tasks is then computed:

$$Rel(T_{\mathcal{T}}, T_t) = 1 - (\frac{Er_t - Er_{\mathcal{T}}}{Er_{\mathcal{T}}}) \tag{4.2}$$

Note that the relatedness value is not symmetric. Applying this to every previous task, we get a relatedness value to each previous task.

We exploit task relatedness in two ways. First, we use it to select the most related task to be used as a prior model for learning the new task. Second, we exploit the level of task relatedness to determine which transfer method to use: fine-tuning or learning-without-forgetting (LwF) [85]. LwF applies additional regularization term insuring that the output of the model trained on the new task remains close to the previous model output. This is done by keeping a separate output layer for the initial task and adding a new one for the new task. Before training on the new task, the outputs of the previous task layer are recorded given the new task data. During training, these outputs are preserved through the use of the knowledge distillation loss [57] (see Chapter 2, Section 2.5). We found in our experiments that LwF only outperforms fine-tuning when the two tasks are sufficiently related. When this is not the case, enforcing the new model to give similar outputs for the old task may actually hurt performance. Fine-tuning, on the other hand, only uses the previous task parameters as a starting point and is less sensitive to the level of task relatedness. Therefore, we apply a threshold on the task

Algorithm 1 Expert Gate

Training Phase input: expert-models $(E_1, ., E_{\mathcal{T}-1})$, tasks-autoencoders $(A_1, ., A_{\mathcal{T}-1})$, new task $(T_{\mathcal{T}})$, data $(D_{\mathcal{T}})$; output: $E_{\mathcal{T}}$, $A_{\mathcal{T}}$

$A_{\mathcal{T}}$ =train-task-autoencoder $(D_{\mathcal{T}})$
(*rel,rel-val*)=select-most-related-task($D_{\mathcal{T}}$,$A_{\mathcal{T}}$, $\{A_t\}$)
if *rel-val* >*rel-th* **then**
    $E_{\mathcal{T}}$=LwF$(E_{rel}, D_{\mathcal{T}})$
else
    $E_{\mathcal{T}}$=fine-tune$(E_{rel}, D_{\mathcal{T}})$
end if

Test Phase input: x ; output: prediction

a=select-expert$(\{A_t\}, x)$
prediction = activate-expert(E_a, x)

relatedness value to decide when to use LwF and when to fine-tune. Algorithm 1 shows the main steps of our Expert Gate in both training and test phase.

4.4 Experiments

First, we compare our method against various baselines on a set of three image classification tasks (Section 4.4.1). Next, we analyze our gate behavior in more detail on a bigger set of tasks (Section 4.4.2), followed by an analysis of our task relatedness measure (Section 4.4.3). Finally, we test Expert Gate on a video prediction problem (Section 4.4.5).

Implementation details We use the activations of the last convolutional layer of an AlexNet pre-trained on ImageNet as image representation for our autoencoders. We experimented with the size of the hidden layer in the autoencoder, trying sizes of 10, 50, 100, 200 and 500, and found an optimal value of 100 neurons. This is a good compromise between complexity and performance. If the task relatedness is higher than 0.85, we use LwF; otherwise, we use fine-tuning. We use MatConvNet framework [154] in all our experiments in this chapter.

4.4.1 Comparison with Baselines

We start with the sequential learning of three image classification tasks: in order, we train on MIT *Scenes* [117], Caltech-UCSD *Birds* [157] and Oxford *Flowers* [111]. To simulate a scenario in which an agent or robot has some prior knowledge, and is

then exposed to datasets in a sequential manner, we start off with an AlexNet model pre-trained on ImageNet. We compare against the following baselines:

1. `Joint Training`. Assuming all training data are always available, a model is jointly trained (by finetuning an AlexNet model pretrained on ImageNet) on all three tasks together.
2. `Multiple fine-tuned models`. Distinct AlexNet models (pretrained on ImageNet) are finetuned separately, one for each task. At test time, an oracle gate is used, i.e. a test sample is always evaluated by the correct model.
3. `Multiple LwF models`. Distinct models are learned with learning-without-forgetting [85], one model per new task, always using AlexNet pre-trained on ImageNet as previous task. This is again combined with an oracle gate.
4. `A single fine-tuned model`. one AlexNet model (pre-trained on ImageNet) sequentially fine-tuned on each task.
5. `A single LwF model`. LwF sequentially applied to multiple tasks. Each new task is learned with all the outputs of the previous network as soft targets for the new training samples. So, a network (pre-trained on ImageNet) is first trained for T_1 data without forgetting ImageNet (i.e. using the pretrained AlexNet predictions as soft targets). Then, this network is trained with T_2 data, now using ImageNet and T_1 specific layers outputs as soft targets; and so on. While preserving ImageNet outputs might seem unfair, it acts as a great source for knowledge transfer and benefits all the later tasks.

For baselines with multiple models (2 and 3), we rely on an oracle gate to select the right model at test time. So reported numbers for these are upper bounds of what can be achieved in practice. The same holds for baseline 1, as it assumes all previous training data are stored and available. Table 4.1 shows the classification accuracy achieved on the test sets of the different tasks. For our Expert Gate system and for each new task, we first select the most related previous task (including ImageNet) and then learn the new task expert model by transferring knowledge from the most related task model, using LwF or finetuning. For the *Single fine-tuned model* and *Single LwF model*, we also report intermediate results in the sequential learning. When learning multiple models (one per new task), LwF improves over vanilla fine-tuning for Scenes and Birds, as also reported by [85][1]. However, for Flowers, performance

[1]Note these numbers are not identical to [85] but show similar trends. At the time of experimentation, the code for LwF was not available, so we implemented this ourselves in consultation with the authors of [85], and used parameters provided by them.

Method	Scenes	Birds	Flowers	Avg.
`Joint Training*`	63.1%	58.5%	85.3%	68.9%
`Multiple fine-tuned models**`	63.4%	56.8%	85.4%	68.5%
`Multiple LwF models**`	63.9%	58.0%	84.4%	68.7%
`Single fine-tuned model`	63.4%	-	-	-
	50.3%	57.3%	-	-
	46.0%	43.9%	84.9%	58.2%
`Single LwF model`	63.9%	-	-	-
	61.8%	53.9%	-	-
	61.2%	53.5%	83.8%	66.1%
`Expert Gate (ours)`	63.5%	57.6%	84.8%	68.6%

Table 4.1: Classification accuracy for the sequential learning of 3 image classification tasks. Methods with * assume all previous training data is still available, while methods with ** use an oracle gate to select the proper model at test time.

degrades compared to fine-tuning. We measure a lower degree of task relatedness to ImageNet for Flowers than for Birds or Scenes (see Figure 4.3) which might explain this effect. Comparing the `Single fine-tuned model` (learned sequentially) with the `Multiple fine-tuned models`, we observe an increasing drop in performance on older tasks: sequentially fine-tuning a single model for new tasks shows catastrophic forgetting and is not a good strategy for continual learning. The `Single LwF model` is less sensitive to forgetting on previous tasks. However, it is still inferior to training exclusive models for those tasks (`Multiple fine-tuned / LwF models`), both for older as well as newer tasks. Lower performance on previous tasks is because of a buildup of errors and degradation of the soft targets of the older tasks. This results in LwF failing to compensate for forgetting in a sequence involving more than 2 tasks. This also adds noise in the learning process of the new task. Further, the previous tasks have varying degree of task relatedness.

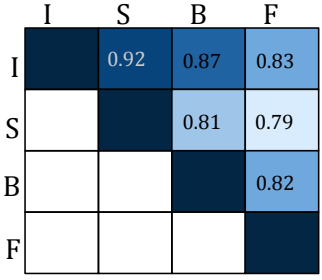


Figure 4.3: Task relatedness. First letters indicate tasks.

On these datasets, we systematically observed the largest task relatedness values for ImageNet (see Figure 4.3). Having to remember other tasks prevents the new task from getting the same benefit of ImageNet as in the `Multiple LwF models` setting. Our `Expert Gate` always correctly identifies the most related task, i.e. ImageNet. Based on the relatedness degree, it used LwF for Birds and Scenes, while fine-tuning was used for Flowers. As a result, the best expert models were learned for each task. At test time, our gate mechanism succeeds to select the correct model for 99.2% of the test samples. This leads to superior results to those achieved

Method	Scenes	Birds	Flowers	Cars	Aircraft	Actions	Avg.
`Joint Training*`	59.5%	56.0%	85.2 %	77.4%	73.4%	47.6%	66.5%
`Most confident model`	40.4%	43.0%	69.2%	78.2%	54.2%	8.2%	48.7%
`Expert Gate`	60.4%	57.0%	84.4%	80.3%	72.2%	49.5%	67.3%

Table 4.2: Classification accuracy for the sequential learning of 6 tasks. Method with * assumes all the training data is available.

by the other two sequential learning strategies (`Single fine-tuned model` and `Single LwF model`). We achieve comparable performance on average to the `Joint Training` that has access to all the tasks data. Also, performance is on par with `Multiple fine-tuned models` or `Multiple LwF models` that both assume having the task label for activating the associated model.

4.4.2 Gate Analysis

The goal of this experiment is to further evaluate our Expert Gate ability in successfully selecting the relevant network(s) for a given test image. For this experiment, we add 3 more tasks: Stanford *Cars* dataset [73], FGVC-*Aircraft* dataset [94], and VOC *Actions*. So in total we deal with 6 different tasks: Scenes→Birds→Flowers→Cars→Aircraft →Actions, along with ImageNet that is considered as a generalist model or initial pre-existing model.

We compare again with `Joint Training`, where we fine-tune the ImageNet pre-trained AlexNet jointly on the six tasks assuming all the data is available. We also compare with a setting with multiple fine-tuned models where the model with the maximum score is selected (`Most confident model`). For our Expert Gate, we follow the same regime as in the previous experiment. The most related task is always ImageNet. Based on our task relatedness threshold, LwF was selected for Actions, while Aircraft and Cars were fine-tuned. Table 4.2 shows the results.

Even though the jointly trained model has been trained on all the previous tasks data simultaneously, its average performance is inferior to our Expert Gate system. This can be explained by the negative inductive bias where some tasks negatively affect others, as is the case for Scenes and Cars.

As we explained in the introduction of this chapter, Section 4.1, deploying all models and taking the max score (`Most confident model`) is not an option: for many test samples the most confident model is not the correct one, resulting in poor performance. Additionally, with the size of each expert model around 220 MB and the size of each autoencoder around 28 MB, there is almost an order of magnitude difference in memory requirements.

Method	Scenes	Birds	Flowers	Cars	Aircraft	Actions	Avg.
`Task Classifier` *using all the tasks data*	97.0 %	98.6%	97.9%	99.3%	98.8%	95.5%	97.8%
`Expert Gate` small *no access to the previous tasks data*	94.6%	97.9%	98.6%	99.3%	97.6%	98.1%	97.6%

Table 4.3: Results on discriminating between the 6 tasks (classification accuracy)

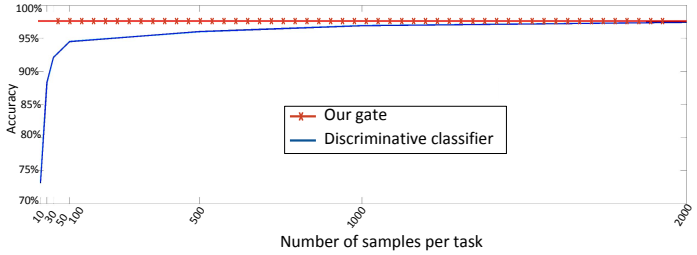


Figure 4.4: Comparison between our gate and the discriminative classifier with varying number of stored samples per task.

Comparison with a discriminative classifier.

We compare with a discriminative classifier trained to predict the task considering the six tasks sequence. For this classifier, we first assume that all data from the previous tasks are stored, even though this is not in line with the task incremental setting. Thus, it serves as an upper bound. For this classifier (`Discriminative Task Classifier`) we use a neural net with one hidden layer composed of 100 neurons (same as our autoencoder code size). It takes as input the same data representation as our autoencoder gate and its output is the different tasks labels. Table 4.3 compares the performance of our gate on recognizing each task data to that of the discriminative classifier. Further, we test the scenario of a discriminative classifier with the number of stored samples per task varying from 10-2000 (Figure 4.4). It approaches the accuracy of our gate with 2000 samples per task. Note that this is $\frac{1}{2}$ to $\frac{1}{3}$ of the size of the used datasets. For larger datasets, an even higher number of samples would probably be needed to match performance. In spite of not having access to any of the previous tasks data, our Expert Gate achieves similar performance to the discriminative classifier. In fact, our Expert Gate can be seen as a sequential classifier with new classes arriving one after another. This is one of the most important results from this chapter: *without ever having simultaneous access to the data of different tasks, our Expert Gate based on autoencoders manages to assign test samples to the relevant tasks equally accurately as a discriminative classifier*.

Analysis of confusion cases

As to have a better understanding of the different kinds of mistakes that we are likely to expect from Expert Gate, we show diverse qualitative examples from the 6 tasks learning sequence.

In Figure 4.5, you can find for each task: an example that has been assigned mistakenly to one of the other tasks. Note that for some cases, the examples shown are the only mistakes made by our Expert Gate. For some test samples even humans have a hard time telling which expert should be activated. Most of these mistakes are explainable

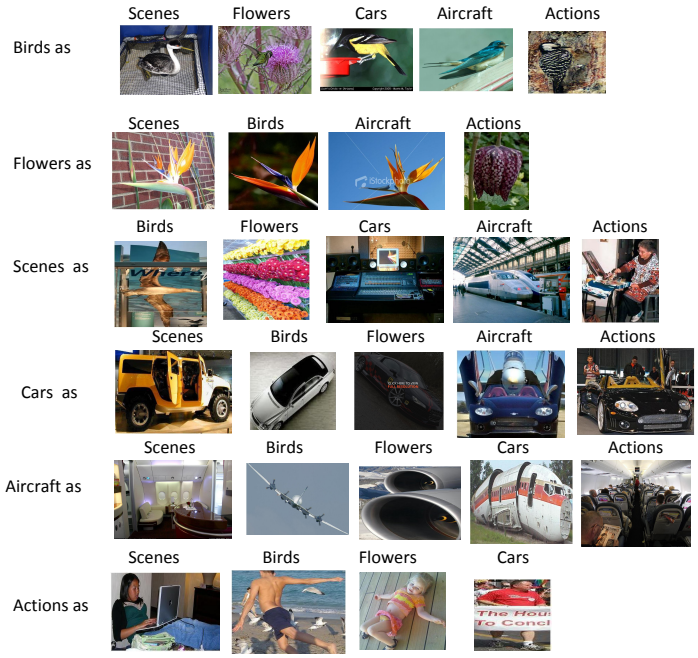


Figure 4.5: Detailed confusion cases that occurred using Expert Gate in the six tasks sequence.

and due to one of the following reasons:

- the image contains objects from two different tasks and our Expert Gate has to choose one of them. For example, Scenes images containing humans can also be classified as Actions. To deal with such cases, it may be preferable in some settings to allow more than one expert to be activated. This can be done

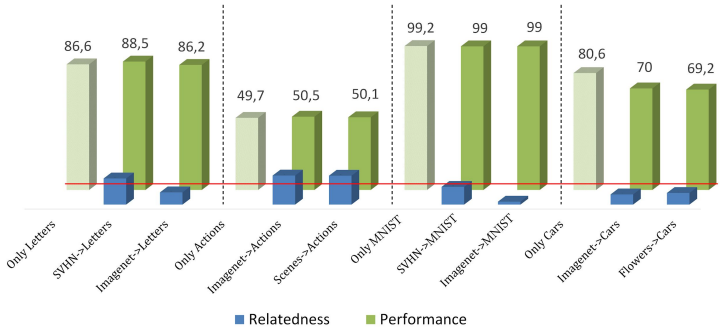


Figure 4.6: Relatedness analysis. The relatedness values are normalized for the sake of better visualization. The red line indicates our relatedness threshold value.

by setting a threshold on the probabilities for the different tasks. We tested this scenario with a threshold of 0.1 and observed 3.7% of the test samples being analyzed by multiple expert models. Note that in this case we can only evaluate the label given by the corresponding task as we are missing the ground truth for the other possible tasks appearing in the image. This leads to an average accuracy of 68.2%, i.e. a further increase of 0.9%.

- the image is an outlier w.r.t. its own task dataset. For example, only a small part of the object appears which means it is not a useful example or it could even harm the classifier if used in the training phase. This sheds light on another potential use of our gate, i.e. to detect outliers. In fact, the autoencoder represents the distribution of each task data. Thus an outlier that is in the long tail of the task distribution will have a higher reconstruction error. This has the potential of being used in cleaning the annotations of each new task data.

- objects from one task that look similar to the images of another task.

4.4.3 Task Relatedness Analysis

In the previous cases, the most related task was always ImageNet. This is due to the similarity between the images of these different tasks and those of ImageNet. Also, the wide diversity of ImageNet classes enables it to cover a good range of these tasks. Does this mean that ImageNet should be the only task to transfer knowledge from, regardless of the current task nature? To answer this question, we add three more different tasks to our previous basket: *SVHN*, *Letters*, and *MNIST*. From the previous set, we pick the two most related tasks, Actions and Scenes, and the two most unrelated tasks, Cars and Flowers. We focus on LwF [85] as a method for knowledge transfer.

We also consider ImageNet as a possible source. We consider the following knowledge transfer cases: Scenes → Actions, ImageNet → Actions, SVHN → Letters, ImageNet → Letters, SVHN → MNIST, ImageNet → MNIST, Flowers → Cars and ImageNet → Cars. Figure 4.6 shows the performance of LwF compared to fine-tuning the tasks with pre-trained AlexNet (indicated by "Only X") along with the degree of task relatedness. The red line indicates the threshold of 0.85 task relatedness used in our previous experiments.

In the case of a high score for task relatedness, LwF uses the knowledge from the previous task and improves performance on the target task – see e.g. (SVHN→Letter, Scenes→Actions, ImageNet→Actions). When the tasks are less related, the method fails to improve and starts to degrade its performance, as in (ImageNet→Letters, SVHN → MNIST). When the tasks are highly unrelated, LwF can even fail to reach a good performance for the new task, as in the case of (ImageNet→ Cars, Flowers→ Cars). This can be explained by the fact that each task is pushing the shared parameters in a different direction and thus the model fails to reach a good local minimum. We conclude that *our gate autoencoder succeeds to predict when a task could help another in the LwF framework and when it cannot*.

4.4.4 Autoencoder Design Choices

We show the effect of using different design choices for our autoencoder gate. As a test case, we use the 3 sequential learning tasks, namely, Scenes, Birds and Flowers. We consider the following alternatives:

- `Linear autoencoder`: no use of nonlinear activation functions. The loss function is the Euclidean distance. This setup learns the same subspace as PCA.
- `No standardization`: the same structure of our autoencoder gate but without the standardization of the input. Also, no fine-tuning is performed here.
- `Sigmoid activation function`: in our design choice, we use a ReLU activation function for the encoding layer. In this baseline, we use the sigmoid activation function for the encoding as well as decoding layer.
- `ReLU activation function`: our design choice of using a ReLU activation function, here without the initialization of the ImageNet pretrained autoencoder.
- `Expert Gate`: our autoencoder gate with the full design: standardization step, ReLU hidden activation and ImageNet fine-tuning.

For all the different alternatives, we use the AdaGrad optimizer [33] for the training of the autoencoder.

Method	Scenes	Birds	Flowers	Avg.
`Linear Autoencoder`	93.6%	98.3%	35.4%	75.8%
`No standardization`	97.4%	97.7%	97.4%	97.5%
`Sigmoid activation function`	97.3%	98.4%	97.4%	97.7%
`ReLU activation function`	97.6%	98.4%	97.4%	97.8%
`Expert Gate`	99.4%	99.2%	99.2%	99.3%

Table 4.4: Comparison of different autoencoder designs: classification accuracy of the autoencoders for the sequential learning of 3 image classification tasks.

Table 4.4 shows the classification accuracy for the task labeling problem achieved by each of the different choices and our Expert Gate autoencoder. It can be noticed that the linear gate (`Linear Autoencoder`) fails to recognize the examples from the Flowers dataset – the linearly learned subspace might not be different from the subspace learned for Birds (due to the visual similarity). There is a slight relative improvement between the gate with the standardization (`ReLU activation function` and `Sigmoid activation function`) and without (`No standardization`). Lastly, our design choice along with finetuning after the autoencoder learned on ImageNet achieves the highest accuracy in recognizing the three different tasks.

4.4.5 Video Prediction

Next, we evaluate our Expert Gate for video prediction in the context of autonomous driving. For video prediction, we use the Dynamic Filter Network (DFN) [67]. Given a sequence of 3 images, the task for the network is to predict the next 3 images. This is quite a structured task, where the task environment and training data affect the prediction results quite significantly. An autonomous vehicle that uses video prediction needs to be able to load the correct model for the current environment. It might not have all the data from the beginning, and so it becomes important to learn specialists for each type of environment, without the need for storing all the training data. Even when all data are available, joint training does not give the best results on each domain, as we show below.

We show experiments conducted on three domains/tasks: for *Highway*, we use the data from DFN [67], with the same train/test split; for *Residential* data, we use the two longest sequences from the KITTI dataset [44]; and for *City* data, we use the Stuttgart sequence from the CityScapes dataset [29], i.e. the only sequence in that dataset with densely sampled frames. We use a 90/10 train/test split on both residential and city datasets. We train the 3 tasks using 3 different regimes: sequential training using a `Single Fine-tuned Model`, `Joint Training` and `Expert Gate`. For video prediction, LwF does not seem applicable as the previous and current task share

Method	Highway	Residential	City	Avg.
`Joint Training*`	14.0	40.7	16.9	23.8
`Single Fine-tuned Model`	13.4	-	-	-
	25.7	45.2	-	-
	26.2	50.0	17.3	31.1
`Expert Gate (ours)`	**13.4**	**40.3**	**16.5**	**23.4**

Table 4.5: Video prediction results (average pixel L1 distance, lower is better). For methods with * all the previous data needs to be available.

the same output space. In this experiment, we use the autoencoders only as gating function. We do not use task relatedness. Video prediction results are expressed as the average pixel-wise ℓ_1-distance between predicted and ground truth images (lower is better), and shown in table 4.5.

Similar trends are observed as for the image classification problem: sequential fine-tuning (`Single Fine-tuned Model`) results in catastrophic forgetting, where a model fine-tuned on a new dataset deteriorates on the original dataset after fine-tuning. `Joint Training` leads to better results on each domain, but requires all the data for training. Our Expert Gate system gives better results compared to both `Single Fine-tuned Model` and `Joint Training`. These experiments show the potential of our Expert Gate system for video prediction tasks in autonomous driving applications.

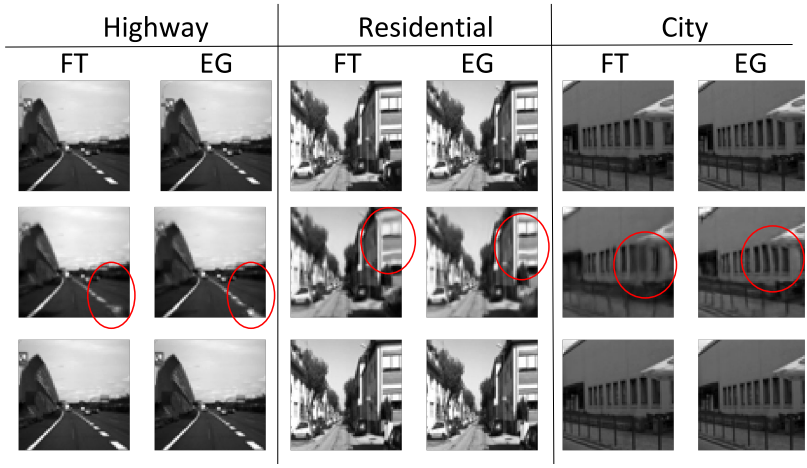


Figure 4.7: Video prediction on the Highway, Residential and City datasets from left to right (two columns for each dataset) using `Single Fine-tuned Model` (first column) and our `Expert Gate` (second column).

Video Prediction - Qualitative Results Figure 4.7 shows qualitative results using sequential fine-tuning and Expert Gate. The first row shows the first in a sequence of 3 images input from the highway, residential and city datasets, to `Single Fine-tuned Model` and `Expert Gate` respectively. The second row contains the last of the three images predicted by the 2 systems, again for the 3 datasets. Ground truth predictions are shown on the last row. It can be seen that `Expert Gate` gives consistently superior qualitative results in the 3 datasets.

4.5 Summary

In the context of incremental task learning, most work has focused on reducing forgetting and on how to exploit knowledge from previous tasks and transfer it to a new task. Little attention has gone to the related and equally important problem of how to select the proper (i.e. most relevant) model at test time. To the best of our knowledge, we have been the first to propose a solution that does not require storing data from previous tasks. Surprisingly, Expert Gate autoencoders can distinguish different tasks equally well as a discriminative classifier trained on all data. Moreover, they can be used to select the most related task and the most appropriate transfer method during training. Combined, this gives us a powerful method for incremental task learning, that outperforms not only the state-of-the-art but also joint training of all tasks simultaneously.

Later research works on continual learning still rarely consider this problem and rely on an imaginary task oracle to decide at test time the correct model/head. In industry, we have been informed through confidential personal communication that our system has been deployed in a big industrial project with over 80 domains of different machine settings.

Our Expert Gate allows for an optimal performance on each learned task without having access to previous data. However, it still learns a new task model and autoencoder per task which leads to a linear increase in storage requirements. In the next chapters, we discuss how to sequentially learn multiple tasks using one model.

Chapter 5

Continual Learning with a Fixed Model Capacity based on Autoencoders

In the previous chapter, we opted for an optimal performance in each seen task while tolerating a linear increase in the number of trained models. We argued that storing the trained models is much cheaper than storing the training datasets and more secure. In this chapter, we consider a different trade-off. We approach continual learning using one fixed capacity training model, again without assuming access to previous tasks data. This requires developing a mechanism that balances the stability-plasticity trade-off explained in Chapter 1. Close to our work described in the previous chapter, we rely on autoencoders. However, here we use them to preserve the knowledge acquired from previous tasks while learning a new one. After each task, an under-complete autoencoder is learned, capturing the features that are crucial for this task achievement. When a new task is presented to the system, we prevent the reconstructions of the features with these autoencoders from changing, which has the effect of preserving the information on which the previous tasks are mainly relying. At the same time, the features are given space to adjust to the most recent environment as only their projection into a low dimensional submanifold is controlled. The proposed system is evaluated on image classification tasks and shows a reduction of forgetting over state of the art at the time of development.

This is a joint work with Amal Rannen. I worked on the initial ideas starting from Expert Gate (see Chapter 4). Then Amal joined and we started working closely together on further developing the approach and deploying it under the task incremental setting.

In the last weeks of this work, I focused on empirical validation while Amal worked on providing theoretical justification for the proposed approach. This work was published as an article in ICCV 2017 [120].

5.1 Introduction

In this chapter, we consider the task incremental setting where tasks associated with their training data are received sequentially and, importantly, no access to previous tasks training data can be gained. In spite of the strict task incremental assumption, this scenario occurs frequently in real applications. For instance, imagine an agent that was trained to localize the defects on a set of factory products. Then, when new products are introduced and the agent has to learn to detect the anomalies in these new products, training images of the previous products might not be available any more and a fine-tuning on the new products images would cause a forgetting of the previous products. Another example is medical imaging applications where user-data is important to be analysed privately. For example, a tumor recognition model might be trained on MRI-images data from one clinic and deployed in different clinics. Now if one clinic wants to improve the model or extend it to other modalities, this must be done without access to the previous data.

The main challenge is to make the learned model adapt to new data from a similar or a different environment [113], without losing knowledge on the previously seen task(s). Most of the classical solutions for this challenge suffer from important drawbacks. Feature extraction (as in [32]), where the model / representation learned for the old task is re-used to extract features from the new data without adapting the model parameters, preserves all previous tasks information but fails to optimally learn the new ones unless tasks are very similar. Fine-tuning (as in [45]), adapts the model to the new task using the optimal parameters of the old task as initialization. As a result, the model is driven towards the newly seen data but forgets what was learned previously. Joint training as we showed in Chapter 4, converges to the best compromise between tasks, but requires the presence of all the data at the same time.

To overcome these drawbacks without the constraint of storing data from the previously seen tasks, yet with a constant model capacity, methods usually impose an additional regularizer to preserve previous tasks knowledge while learning a new one. Such a regularizer is exploited to (i) preserve the performance on the previously seen data, (ii) improve this knowledge using inductive bias [104] from the new task data, and (iii) regularize the training of the new task, which can be beneficial for the performance.

In this chapter we aim at preserving the knowledge of the previous tasks and possibly benefiting from this knowledge while learning a new task, without storing data from previous tasks using a fixed model capacity.

Learning without forgetting (LwF) [85], was the first work to re-introduce the data focused regularization approach after the recent success of neural networks. As explained in Chapter 4, Section 4.4.3, it proposes to preserve the previous performance through the knowledge distillation loss introduced in [57]. It considers a shared convolutional network between the different tasks in which only the last classification layer is task specific. When encountering a new task, the outputs of the existing classification layers given the new task data are recorded. During training, these outputs are preserved through a modified cross-entropy loss that softens the class probabilities in order to give a higher weight to the small outputs. More details about this loss and the method can be found in Section 5.2.2. This method reduces the forgetting, especially when the datasets come from related manifolds. Nevertheless, it has been shown by iCaRL [122] that LwF suffers from a build up of errors in a sequential scenario where the data comes from the same environment. Similarly, we showed in the previous chapter that LwF performance drops when the model is exposed to a sequence of tasks drawn from different distributions. iCaRL [122] proposes to store a selection of the previous tasks data to overcome this issue – something we try to avoid.

In the work presented in this chapter, we propose a compromise between these two methods. Rather than heavily relying on the new task data or requiring to store samples from the previous tasks, we introduce the use of autoencoders as a tool to preserve the knowledge from one task while learning another. For each task, an undercomplete autoencoder is trained after training the task model. It captures the most important features for the task objective. When facing a new task, this autoencoder is used to ensure the preservation of those important features. This is achieved by defining a loss on the reconstructions made by the autoencoder, as we will explain in the following sections. In this manner, we only restrict a subset of the features to be unchanged while we give the model the freedom to adapt itself to the new task using the remaining capacity.

Below, in Section 5.2, we describe how to use the autoencoders to avoid catastrophic forgetting and motivate our choice by a short analysis that relates the proposed objective to the joint training scheme. In Section 5.3, we describe the experiments that we conducted, report and discuss their results before concluding in Section 5.4.

5.2 Overcoming Forgetting with Autoencoders

When training one model on a sequence of tasks, the best compromise performance between all the tasks simultaneously is achieved when the network is trained on the data from all the considered tasks at the same time (as in joint training). This performance is of course limited by the capacity of the used model, and can be considered an upper

bound to what can be achieved in an task incremental setting, where the data of previous tasks are no longer accessible when learning a new one.

5.2.1 Joint Training

In the following, we will use Q_t to denote the distribution from which the dataset D_t of the task T_t is sampled, and $x_n^{(t)}$ and $y_n^{(t)}$ for the data samples (input and output labels). When we have access to the data from all $\mathcal{T}$ tasks jointly, the network training aims to control the statistical risk:

$$\sum_{t=1}^{\mathcal{T}} \mathbb{E}_{Q_t(x^{(t)},y^{(t)})}[\ell(f_t(x^{(t)}), y^{(t)})], \tag{5.1}$$

by minimizing the empirical risk:

$$\sum_{t=1}^{\mathcal{T}} \frac{1}{N_t} \sum_{n=1}^{N_t} \ell(f_t(x_n^{(t)}), y_n^{(t)}), \tag{5.2}$$

where N_t is the number of samples and f_t the function implemented by the network for task t. For most of the commonly used models, we can decompose f_t as $O_t \circ O \circ F$ where:

- F is a feature extraction function (e.g. Convolutional layers in ConvNets)
- $O_t \circ O$ is a task operator. It can be for example a classifier or a segmentation operator. O is shared among all tasks, while O_t is task specific. (e.g. in AlexNet, O_t could be the last fully-connected layer, and O the remaining fully-connected layers.)

The upper part of Figure 5.1 gives a scheme of this general model. For simplicity, we will focus below on two-task training before generalizing to a multiple task scenario in section 5.2.4.

5.2.2 Shortcomings of Learning without Forgetting

As a first step, we want to understand the limitations of LwF [85]. In that work, it is suggested to replace in Equation 5.1 $\ell(O_1 \circ O \circ F(x^{(1)}),\ y^{(1)})$ with $\ell(O_1 \circ O \circ F(x^{(2)}), O_1^* \circ O^* \circ F^*(x^{(2)}))$, where $O_1^* \circ O^* \circ F^*$ is obtained from training the network on the first task. If we suppose that the model has enough capacity to integrate

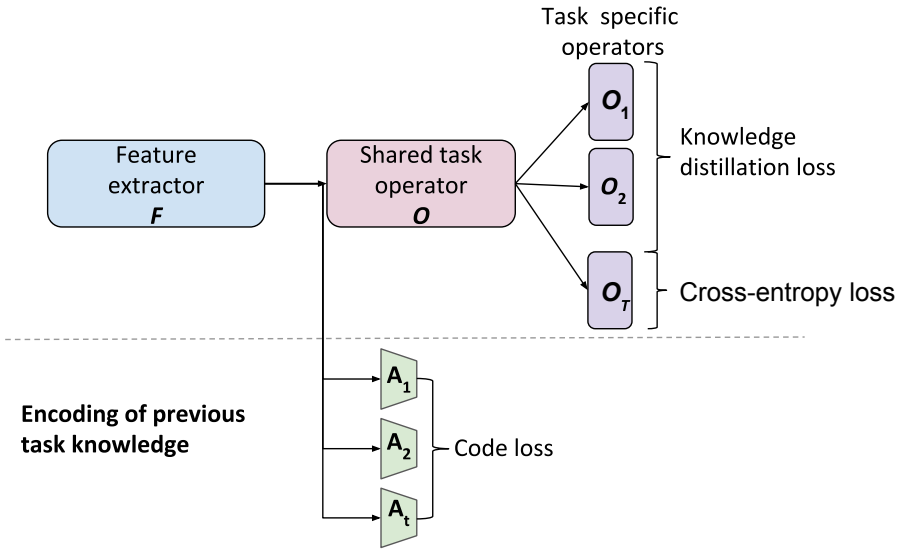


Figure 5.1: Diagram of the proposed model. Above the dotted line are the model components that are retained during test time, while below the dashed line are components necessary to our improved training scheme.

the knowledge of the first task with a small generalization error, then we can consider that

$$\mathbb{E}_{Q_1(x)}[\ell(O_1 \circ O \circ F(x^{(1)}), O_1^* \circ O^* \circ F^*(x^{(1)}))] \tag{5.3}$$

is a reasonable approximation of $\mathbb{E}_{Q_1(x^{(1)},y^{(1)})}[\ell(O_1 \circ O \circ F(x^{(1)}), y^{(1)})]$. However, in order to be able to compute the measure in Equation 5.3 using samples from $x^{(2)}$, further conditions need to be satisfied.

If we consider that $O_1 \circ O \circ F$ tries to learn an encoding of the data in the target distribution $Q_1(x^{(1)})$, then one can say that the loss of information generated by the use of $Q_2(x^{(2)})$ instead of $Q_1(x^{(1)})$ is a function of the Kullback-Leibler divergence of the two related probability distributions.

In this work, we build on top of the LwF method. In order to make the used approximation less sensitive to the data distributions, we see an opportunity in controlling $\|O_1 \circ O \circ F(x^{(1)}) - O_1 \circ O \circ F(x^{(2)})\|$. Under mild conditions about the model functions, namely Lipschitz continuity, this control allows us to use $O_1 \circ O \circ F(x^{(2)})$ instead of $O_1 \circ O \circ F(x^{(1)})$ to better approximate the first task loss in Equation 5.1. Note that the condition of continuity on which this observation is

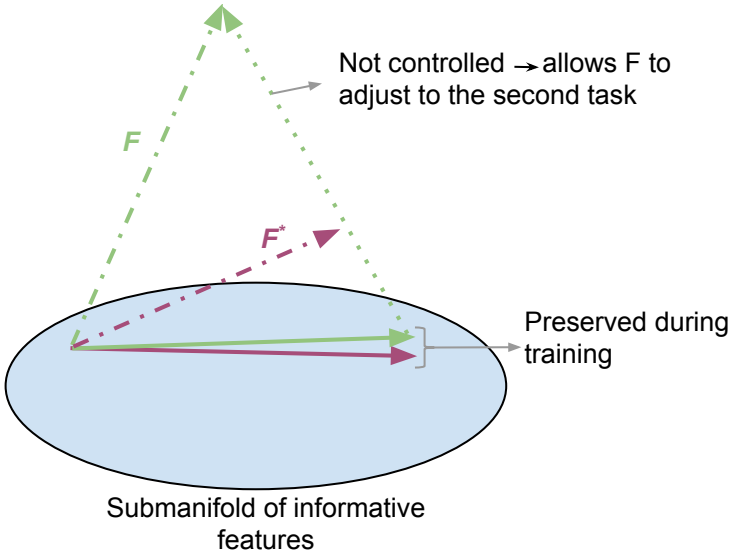


Figure 5.2: Preservation of the features that are important for task 1 while training on task 2. During training, we enforce the projection of F into the submanifold that captures these important features to stay close to the projection of F^*, the optimal features for the first task. The part of F that is not meaningful for the first task is allowed to adjust to the variations of the second task.

based is not restrictive in practice. Indeed, most of the commonly used functions in deep models satisfy this condition (e.g. sigmoid, ReLU).

Our main idea is to learn a submanifold of the representation space $F(x^{(1)})$ that contains the most informative features for the first task. Once this submanifold is identified, if the projections of the features $F(x^{(2)})$ onto this submanifold do not change much during the training of a second task, then two consequences follow: (i) $F(x^{(2)})$ will stay informative for the first task during the training, and (ii) at the same time there is room to adjust to the second task as only its projection in the learned submanifold is controlled. Figure 5.2 gives a simplified visualization of this mechanism. In the next section, we propose a method to learn the submanifold of informative features for a given task using autoencoders.

5.2.3 Informative Features Preservation

When beginning to train the second task, the feature extractor F^* of the model is optimized for the first task. A feature extraction type of approach would keep this

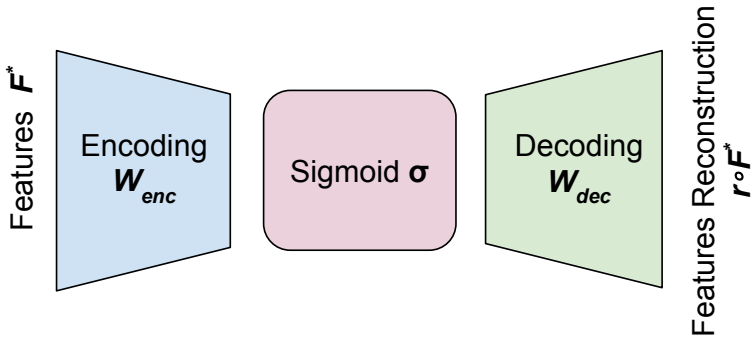


Figure 5.3: Scheme of an undercomplete autoencoder trained to capture the important features submanifold.

operator unchanged in order to preserve the performance on the previous task. This is, however, overly conservative, and usually suboptimal for the new task. Rather than preserving *all* the features during training, our main idea is to preserve only the features that are the most informative for the first task while giving more flexibility for the other features in order to improve the performance on the second task. An autoencoder [21] trained on the representation of the first task data obtained from an optimized model can be used to capture the most important features for this task.

Learning the informative submanifold with Autoencoders

An under-complete autoencoder, where the dimension of the code is smaller than the dimension of the input, learns a lower dimensional projection of its input (see also Chapter 2, Section 2.2). The learned submanifold represents best the structure of the input data. More precisely, we choose to use a two-layer network with a sigmoid activation in the hidden layer: $r(x) = W_{dec}\sigma(W_{enc}x)$. We want the encoding layer to act as a switch where non informative features have a close to zero projection and informative ones are preserved with close to one projection, hence we favored the use of a sigmoid activation function over the ReLU activation function used in Chapter 4. Figure 5.3 shows a general scheme of such an autoencoder.

Here, our aim is to obtain through the autoencoder a submanifold that captures the information that is not only important to reconstruct the features (output of the feature extraction operator F^*) of the first task, but also important for the task operator

$(O_1^* \circ O^*)$. We therefore select our training objective as:

$$\arg\min_r \mathbb{E}_{Q_1(x^{(1)},y^{(1)})}[\beta\|r(F^*(x^{(1)})) - F^*(x^{(1)})\|_2 + \ell(O_1^* \circ O^*(r(F^*(x^{(1)}))), y^{(1)})], \tag{5.4}$$

where ℓ is the loss function used to train the model on the first task data. β is a hyper-parameter that controls the compromise between the two terms in this loss. In this manner, the autoencoder represents the variations that are needed to reconstruct the input and at the same time contain the information that is required by the task operator.

Representation control with separate task operators

To explain how we use these autoencoders, we start with the simple case where there is no task operator shared among all tasks (i.e., $O = \emptyset$). The model is then composed of a common feature extractor F, and a task specific operator for each task O_t. In a two tasks scenario, after training the first task, we have O_1^* and F^* optimized for that task. Then, we train an undercomplete autoencoder using $F^*(x_n^{(1)})$ minimizing the empirical risk corresponding to Equation 5.4. The optimal performance for the first task, knowing that the operator O_1 is kept equal to O_1^*, is obtained with F equal to F^*.

Nevertheless, preventing F from changing will lead to suboptimal performance on the second task. The idea here is to keep only the projection of F into the manifold represented by the autoencoder $(r \circ F)$ unchanged. The second term of Equation 5.4 explicitly enforces r to represent the submanifold needed for good performance on T_1. Thus, controlling the distance between $r \circ F$ and $r \circ F^*$ will preserve the necessary information for T_1. Since this autoencoder captures the submanifold of the features that is crucial to the task operator T_1, preserving the projection of F would prevent the performance of the model on the first task from dropping. From the undercompleteness of the encoder, r projects the features into a lower dimensional manifold, and by controlling only the distance between the reconstructions, we give the features flexibility to adapt to the second task variations. Thus, the autoencoder can be used to realize the mechanism described in Figure 5.2.

Representation control with shared task operator

We now consider the model presented in Figure 5.1 where a part of the task operator is shared among the tasks as in the setting used in LwF [85]. Our main idea is to start from the loss used in the LwF method and add an additional term coming from the idea presented previously in this section. Thus, in a two tasks scenario, in addition to the loss used for the second task, we propose to use two constraints:

1. The first constraint is the knowledge distillation loss (ℓ_{dist}) used in [85]. If $\hat{y} = O_1 \circ O \circ F(x^{(2)})$ and $y^* = O_1^* \circ O^* \circ F^*(x^{(2)})$ then:

$$\ell_{dist}(\hat{y}, y^*) = -\langle z^*, \log \hat{z} \rangle \tag{5.5}$$

   where $\log$ is operated entry-wise and

$$z_i^* = \frac{y_i^{*1/\tau}}{\sum_j y_j^{*1/\tau}} \text{ and } \hat{z}_i = \frac{\hat{y}_i^{1/\tau}}{\sum_j \hat{y}_j^{1/\tau}} \tag{5.6}$$

   The application of a high temperature τ increases the small values of the output and reduces the weight of the high values. This mitigates the influence of the use of different data distributions.

2. The second constraint is related to the preservation of the reconstructions of the second task features ($r \circ F^*(x^{(2)})$). The goal of this constraint is to keep $r \circ F$ close to $r \circ F^*$ as explained previously.

For the second constraint, rather than controlling the distance between the reconstructions, we will here constrain the codes $\sigma(W_{enc}\cdot)$. From sub-multiplicity of the Frobenius norm, we have:

$$\|r(x_1) - r(x_2)\|_2 \leq \|W_{dec}\|_F \|\sigma(W_{enc}x_1) - \sigma(W_{enc}x_2)\|_2.$$

The advantage of using the codes is their lower dimension. As the codes or reconstructions need to be recorded before beginning the training on the second task, using the codes will result in a better usage of the memory.

Finally, this results in minimizing the following objective function for the training of the second task:

$$\begin{aligned} J = \mathbb{E}_{Q_2(x)} \Big[& \ell(O_2 \circ O \circ F(x^{(2)}), y^{(2)})) \\ & + \ell_{dist}(O_1 \circ O \circ F(x^{(2)}), O_1^* \circ O^* \circ F^*(x^{(2)})) \\ & + \frac{\alpha}{2} \|\sigma(W_{enc}F(x^{(2)})) - \sigma(W_{enc}F^*(x^{(2)}))\|_2^2 \Big]. \end{aligned} \tag{5.7}$$

The choice of the parameter α will be done through model selection.

5.2.4 Training Procedure

The proposed method in Section 5.2.3 generalizes easily to a sequence of tasks. An autoencoder is then trained after each task. Even if the needed memory will grow

Algorithm 2 Encoder based Lifelong Learning (EBLL)

Input :
F^* shared feature extractor; O^* shared task operator; $\{O_t\}_{t=1..\mathcal{T}-1}$ previous task operators;
$\{W_{enc,t}\}_{t=1..\mathcal{T}-1}$ previous task encoders;
$(X^{(\mathcal{T})}, Y^{(\mathcal{T})})$ training data and ground truth of the new task $\mathcal{T}$;
α_t and β hyper parameters
Initialization :
1: **for** $t \in \{1, \mathcal{T}-1\}$ **do**
2: $Y_t^* = O_t^* \circ O^* \circ F^*(X^{(\mathcal{T})})$ //record task targets
3: $C_t^* = \sigma(W_{enc,t} F^*(X^{(\mathcal{T})}))$ //record new data codes
4: **end for**
5: $O_{\mathcal{T}} \leftarrow$ Init($|Y^{(\mathcal{T})}|$) // initialize new task operator
Training :
6: $O_t^*, O^*, F^* \leftarrow \arg\min_{O_t, O, F} \Big[\ell(O_{\mathcal{T}} \circ O \circ F(X^{(\mathcal{T})}), Y^{(\mathcal{T})})$
$+ \sum_{t=1}^{\mathcal{T}-1} \ell_{dist}(O_t \circ O \circ F(X^{(\mathcal{T})}), Y_t^*) + \sum_{t=1}^{\mathcal{T}-1} \frac{\alpha_t}{2} \|\sigma(W_{enc,t} F(X^{(\mathcal{T})})) - C_t^*\|_2^2\Big]$
7: $(W_{enc,\mathcal{T}}, W_{dec,\mathcal{T}}) \leftarrow$ autoencoder($O_{\mathcal{T}}^*, O^*, F^*, X^{(\mathcal{T})}, Y^{(\mathcal{T})}; \beta$) // minimizes Equation 5.4

linearly with the number of tasks, the memory required by an autoencoder is a small fraction of that required by the global model. For example, in the case of AlexNet as a base model, an autoencoder comprises only around 1.5% of the memory. Figure 5.1 displays the model we propose to use.

In practice, at task $T_{\mathcal{T}}$ the empirical risk defined by the objective function in Equation 5.8 is minimized.

$$
\begin{aligned}
J = \frac{1}{N_{\mathcal{T}}} \sum_{n=1}^{N_{\mathcal{T}}} \Bigg(& \ell(O_{\mathcal{T}} \circ O \circ F(x_n^{(\mathcal{T})}), y_n^{(\mathcal{T})}) \\
& + \sum_{t=1}^{\mathcal{T}-1} \ell_{dist}(O_t \circ O \circ F(x_n^{(\mathcal{T})}), O_t^* \circ O^* \circ F^*(x_n^{(\mathcal{T})})) \\
& + \sum_{t=1}^{\mathcal{T}-1} \frac{\alpha_t}{2} \|\sigma(W_{enc,t} F(x_n^{(\mathcal{T})})) - \sigma(W_{enc,t} F^*(x_n^{(\mathcal{T})}))\|_2^2 \Bigg).
\end{aligned} \tag{5.8}
$$

The training is done using stochastic gradient descent (SGD) [20]. The autoencoder training is also done by SGD but with an adaptive gradient method, AdaDelta [166] which alleviates the need for setting the learning rates and has nice optimization properties. Algorithm 2 shows the main steps of the proposed method.

5.3 Experiments

We compare our method against several baselines on image classification tasks. We consider sets of 2, 3 and 5 tasks learned sequentially, in two settings: 1) when the first task is a large dataset, and 2) when the first task is a small dataset.

Architecture Similarly to the experiments performed in the previous chapter, we deploy AlexNet [76] as a backbone architecture due to its widespread use and similarity to other popular architectures. The feature extraction block F corresponds to the convolutional layers. By default, the shared task operator O corresponds to all but the last fully connected layers (i.e., $fc6$ and $fc7$), while the task-specific part O_t contains the last classification layer ($fc8$). Other choices for F and O are possible. During the training, we used an α of 10^{-3} for ImageNet and 10^{-2} for the rest of the tasks. Note that this parameter sets the trade off between the allowed forgetting on the previous task and the performance on the new task.

For the autoencoders, we use a very shallow architecture, to keep their memory footprint low. Both the encoding as well as the decoding consist of a single fully connected layer, with a sigmoid as non-linearity in between. The dimensionality of the codes is 100 for all datasets, except for ImageNet where we use a code size of 300. The size of the autoencoder is 3MB, compared to 250MB for the size of the network model. The training of the autoencoders is done using AdaDelta as explained in Section 5.2.4. During training of the autoencoders, we use a hyperparameter β (cf. Equation 5.4) to find a compromise between the reconstruction error and the classification error. This parameter is tuned manually and is set to 10^{-6} in all cases.

Datasets We use multiple datasets of moderate size: MIT *Scenes* [117], Caltech-UCSD *Birds* [157], Oxford *Flowers* [111] and VOC *Actions* [36].

For the scenario based on a large initial dataset, we start from ImageNet [133], which has more than 1 million training images. For the small dataset scenario, we start from Oxford *Flowers*, which has only 2,040 training and validation samples.

The reported results are obtained with respect to the test sets of Scenes, Birds, Flowers and Actions, and on the validation set of ImageNet. As in LwF [85], we need to record the targets corresponding to the old tasks before starting the training procedure for a new task. Here, we perform an offline augmentation with 10 variants of each sample (different crops and flips).

Compared Methods We compare our method with Learning without Forgetting (`LwF`) [85], which represents the state-of-the-art at the time of development (2017).

Additionally, we consider two baselines: `Finetune`, where each model (incl. F and O) is learned for the new task using the previous task model as initialization, and `Feature extraction`, where the weights of the previous task model (F and O) are fixed and only the classification layer (O_t) is learned for each new task. We coin our method (`EBLL`) as short for Encoder Based Lifelong Learning which is the name of the published article. We also report results for a variant of our method, `EBLL-separateFCs` where we only share the representation layers (F) while each task has its own fully connected layers (i.e., $O = \emptyset$ and $O_t = \{fc6 - fc7 - fc8\}$). This variant aims at finding a universal representation for the current sequence of tasks while allowing each task to have its own fully connected layers. With less sharing, the risk of forgetting is reduced, at the cost of a higher memory consumption and less regularization for new tasks. Note that in this case the fully connected layers of the previous tasks are not retrained, and thus there is no need to use the knowledge distillation loss. Moreover, task autoencoders can be used at test time to activate only the fully connected layers of the task that a test sample belongs to, in a similar manner to what we did in the previous chapter.

Setup We consider sequences of 2, 3 and 5 tasks. In the **Two Tasks** setup, we are given a model trained on one previously seen task and then add a second task to learn. This follows the experimental setup of LwF [85]. In their work, all the tested scenarios start from a large dataset, ImageNet. Here we also study the effect of starting from a small dataset, Flowers.[1]

Further, we also consider a setup involving **Three Tasks**. First, we use a sequence of tasks starting from ImageNet, i.e. ImageNet $\rightarrow$ Scenes $\rightarrow$ Birds. Additionally, we consider Flowers as a first task in the sequence Flowers $\rightarrow$ Scenes $\rightarrow$ Birds. Note that this is different from what was conducted in [85] where the sequences were only composed of splits of one dataset i.e. one task overall. Finally, for a stronger validation of our method, we also test on a longer sequence: ImageNet $\rightarrow$ Scenes $\rightarrow$ Birds $\rightarrow$ Flowers $\rightarrow$ Actions.

Results Table 5.1 shows, for the different compared methods, the achieved performance on the Two Tasks scenario with ImageNet as the first task. While `Finetune` is optimal for the second task, it shows the most forgetting of the first task. The performance on the second task is on average comparable for all methods except for `Feature extraction`. Since the `Feature extraction` baseline doesn't allow the weights of the model to change and only optimizes the last fully connected layers, its performance on the second task is suboptimal and significantly lower than

[1] Due to the small size of the Flowers dataset, we use a network pretrained on ImageNet as initialization for training the first task model. The main difference hence lies in the fact that in this case we do not care about forgetting ImageNet.

	ImageNet → Scenes		ImageNet → Birds		ImageNet → Flowers		Average	
	Acc. on T_1	Acc. on T_2	Acc. on T_1	Acc. on T_2	Acc. on T_1	Acc. on T_2	Forg. on T_1	Drop on T_2
`Finetune`	48.0 (-9)	65.0 (ref)	41.3 (-15.7)	59.0 (ref)	50.8 (-6.2)	86.4 (ref)	10.3	0
`Feature extraction`	57.0 (ref)	60.6 (-4.4)	57.0 (ref)	51.6 (-7.4)	57.0 (ref)	84.6 (-1.8)	0	4.5
`LwF`	55.4 (-1.6)	65.0 (-0)	54.4 (-2.6)	58.9 (-0.1)	55.6 (-1.4)	85.9 (-0.5)	1.9	0.2
`EBLL`	56.3 (-0.7)	64.9 (-0.1)	55.3 (-1.7)	58.2 (-0.8)	56.5 (-0.5)	86.2 (-0.2)	1.0	0.4
`EBLL separate FCs`	57.0 (-0)	65.9 (+0.9)	57.0 (-0)	57.7 (-1.3)	56.5 (-0.5)	86.4 (-0)	0.2	0.1

Table 5.1: Classification accuracy (%) for the Two Tasks scenario starting from ImageNet. For the first task, the reference performance is given by Feature extraction. For the second task, we consider `Finetune` as the reference as it is the best that can be achieved by one task alone.

	Flowers → Scenes		Flowers → Birds		Average	
	Acc. on T_1	Acc. on T_2	Acc. on T_1	Acc. on T_2	Forg. on T_1	Drop on T_2
`Finetune`	61.6 (-24.8)	63.9 (ref)	66.6 (-19.8)	57.5 (ref)	22.3	0
`Feature extraction`	86.4 (ref)	59.6 (-4.3)	86.4 0	48.6 (-8.9)	0	6.6
`LwF`	83.7 (-2.7)	62.2 (-1.7)	82.0 (-4.4)	52.2 (-5.3)	3.6	3.5
`EBLL`	84.9 (-1.5)	62.3 (-1.6)	83.0 (-3.4)	52.0 (-5.5)	2.4	3.5
`EBLL-separateFCs`	86.4 (-0)	63.0 (-0.9)	85.4 (-1.0)	55.1 (-2.4)	0.5	1.6

Table 5.2: Classification accuracy (%) for the Two Tasks scenario starting from Flowers. For the first task, the reference performance is given by Feature extraction. For the second task, we consider `Finetune` as reference as it is the best that can be achieved by one task alone.

	ImageNet	Scenes	Birds	Avg. Accuracy	Avg. Forgetting
`Finetune`	37.5%	45.6%	**58.1**%	47.2%	13.0%
`LwF`	53.3%	63.5%	57.2 %	58.0%	1.7%
`EBLL`	**54.9**%	**64.7**%	56.9%	**58.8**%	**1.3**%

Table 5.3: Classification accuracy for the Three Tasks scenario starting from ImageNet. `EBLL` achieves the best trade off between the tasks in the sequence with less forgetting to the previous tasks.

the other methods. Naturally, the performance of the previous task is kept unchanged in this case. `EBLL-separateFCs` shows high performance on both tasks, with the performance of the second task being comparable or better to the methods with shared FCs. This variant of our method has a higher capacity as it allocates separate fully connected layers for each task, yet its memory consumption increases more rapidly as tasks are added, a severe drawback. Our method with a complete shared model `EBLL` systematically outperforms the `LwF` method on the previous task and on average achieves a similar performance on the second task.

	Flowers	Scenes	Birds	Avg. Accuracy	Avg. Forgetting
Finetune	51.2%	48.1%	**58.5%**	51.6%	17.8%
LwF	81.1%	59.1%	52.3%	64.1%	2.8%
EBLL	**82.8%**	**61.2 %**	51.2%	**65.0%**	**1.6%**

Table 5.4: Classification accuracy for the Three Tasks scenario starting from Flowers. EBLL achieves the best trade off between the tasks in the sequence with less forgetting to the previous tasks.

When we start from a smaller dataset, Flowers, the same trends can be observed, but with larger differences in accuracy (Table 5.2). The performance on the second task is lower than that achieved with ImageNet as a starting point for all the compared methods. This is explained by the fact that the representation obtained from ImageNet is more meaningful for the different tasks than what has been finetuned for Flowers. Differently from the ImageNet starting case, EBLL-separateFCs achieves a considerably better performance on the second task than EBLL and LwF while preserving the previous task performance. Finetune shows the best performance on the second task while suffering from severe forgetting on the previous task. The pair of tasks here is of a different distribution and finding a compromise between the two tasks is a challenging problem. As in the previous case, EBLL reduces the forgetting of LwF while achieving a similar average performance on the second task.

Overall, EBLL-separateFCs achieves the best performance on the different pairs of tasks. However, it requires allocating separate fully connected layers for each task which requires a lot of memory. Thus, for the sequential experiments we focus on the shared model scenario.

In Table 5.3 we report the performance achieved by EBLL, LwF and Finetune for the sequence of ImageNet $\rightarrow$ Scenes $\rightarrow$ Birds. As expected, the Finetune baseline suffers from severe forgetting on the previous tasks. The performance on ImageNet (the first task) drops from 57% to 37.9% after finetuning on the third task. As this baseline does not consider the previous tasks in its training procedure, it has the advantage of achieving the best performance on the last task in the sequence.

EBLL continually reduces forgetting compared to LwF on the previous tasks while showing a comparable performance on the new task in the sequence. For example, EBLL achieves 54.9% on ImageNet compared to 53.3% by LwF. Similar conclusions can be drawn regarding the sequential scenario starting from Flowers as reported in Table 5.4.

Effect of the code length In order to examine the effect of varying the code size, we apply our method to the two-task experiment ImageNet $\rightarrow$ Scenes using AlexNet and autoencoders with code size varying from 20 to 9216 (full feature dimension)

trained on the output of $conv5$. Figure 5.4 shows the classification accuracies for both of the datasets for different code sizes. The results indicate that the smaller code sizes favor the performance on the new task, while the larger code sizes lean towards a better preservation of the old task knowledge but a lower performance on the new task. This experiment confirms the motivation behind using autoencoders to encode the knowledge of previous tasks: a larger code means then a better preservation (see Figure 5.2 and Section 5.2.3), but a very large code preserves a noisy information, which explains the drop in the first task performance for the largest codes.

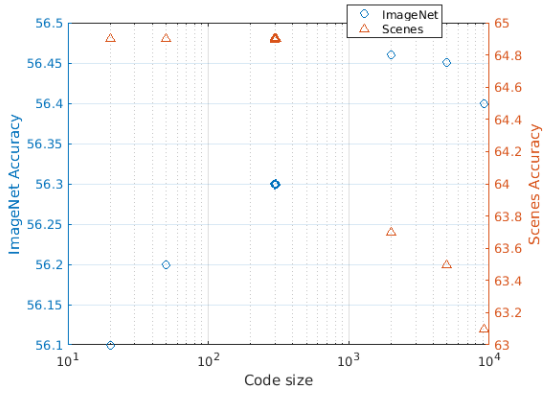


Figure 5.4: Classification accuracy for the Two Tasks scenario ImageNet → Scenes with different code sizes. The points in bold mark the code size corresponding to the setting of the experiments reported in Table 5.1.

Performance on longer sequences We compare the performance of our method to LwF on a longer sequence of five tasks: ImageNet → Scenes → Birds → Flowers → Actions. Figure 5.5 shows the obtained accuracies for the five datasets with both methods. It shows that our method outperforms LwF by 1.32% on average over all the tasks. The performance of our method on ImageNet after five tasks (53.6%) is higher than the performance of LwF (51.2%) . It is worth noticing that our performance after 5 tasks is better than LwF performance after only 3 tasks (53.3%, Table 5.3).

5.4 Summary

Strategies for efficient continual learning are still an open research problem. In this chapter, we tackled the problem of learning a sequence of tasks using only the data from the most recent environment, based on a fixed capacity training model while aiming at obtaining a reasonable performance on the whole sequence.

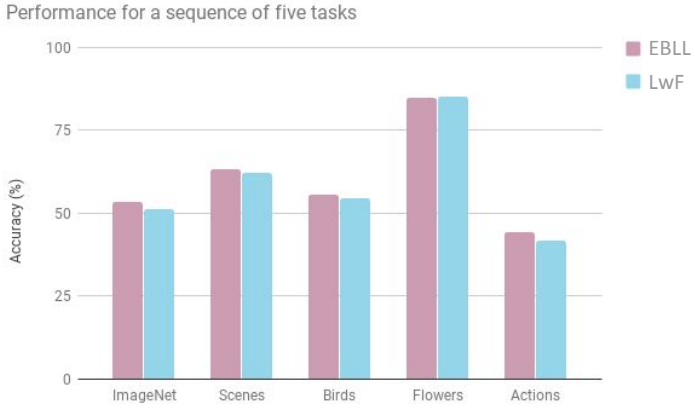


Figure 5.5: Classification accuracy for the Five Tasks scenario. The tasks (horizontal axis) are presented sequentially from left to right. Figure compares the accuracy of each task at the end the sequence for both ours (`EBll`) and `LwF`.

The solution presented here reduces forgetting of earlier tasks by controlling the distance between the representations of the different tasks. We suggest to preserve the features that are crucial for the performance in the corresponding environments. Undercomplete autoencoders are used to learn the submanifold that represents these important features. The method is tested on image classification problems, in sequences of two, three or five tasks, starting either from a small or a large dataset. An improvement in performance over the state-of-the-art at time of development is achieved in all the tested scenarios. Especially, we showed a better preservation of the old tasks.

Despite the demonstrated improvements using a fixed capacity training model, the solution proposed in this chapter still requires for a small but linear increase in memory. After each task, we need a post processing phase to train the autoencoder and a further preprocessing phase for recording the targets on the new task data. In the next chapter, we present an alternative strategy that moves a step forward towards a smoother one model continual learning solution.

Chapter 6

Importance Weight Regularization

Humans can learn in a continuous manner. Old rarely utilized knowledge can be overwritten by new incoming information while important, frequently used knowledge is prevented from being erased. Continual learning so far has focused mainly on accumulating knowledge over tasks and overcoming forgetting of all seen tasks. In this chapter, we argue that, given the limited model capacity and the unlimited new information to be learned, knowledge has to be preserved or erased selectively. Inspired by neuroplasticity, we propose a novel approach for continual learning, coined Memory Aware Synapses (`MAS`). It computes the importance of the parameters of a neural network in an unsupervised and online manner. Given a new sample which is fed to the network, `MAS` accumulates an importance measure for each parameter of the network, based on how sensitive the predicted output is to a change in this parameter. When learning a new task, changes to important parameters can then be penalized, effectively preventing important knowledge related to previous tasks from being overwritten. Further, we show an interesting connection between a local version of our method and Hebb's rule, which is a model for the learning process in the brain. We test our method on a sequence of object recognition tasks and on the challenging problem of learning an embedding for predicting <subject, predicate, object> triplets. We show state-of-the-art performance at the time of development (2018) and, for the first time, the ability to adapt the importance of the parameters based on unlabeled data towards what the network needs (not) to forget, which may vary depending on test conditions.

This work was the fruit of collaboration with Francisca Babiloni and researchers from Facebook AI Research (FAIR), Mohamed Elhoseiny and Marcus Rohrbach. It was published as an article in ECCV 2018 [5].

Figure 6.1: Our considered learning setup. As common in the task incremental setting, tasks are learned in sequence, one after the other. If, in between learning tasks, the agent is active and performs the learned tasks, we can use these unlabeled samples to update importance weights for the model parameters. Data that appears frequently, will have a bigger contribution. This way, the agent learns what is important and should not be forgotten.

6.1 Introduction

When we started looking into this problem (2017), continual learning methods had mostly (albeit not exclusively) been applied to relatively short sequences – often consisting of no more than two tasks (e.g. [83, 85, 120]), and using relatively large networks with plenty of capacity (e.g. [6, 38, 134]). However, in a true continual learning setting with a never-ending list of tasks, the capacity of the model sooner or later reaches its limits and compromises need to be made. Instead of aiming for no forgetting at all, figuring out what can possibly be forgotten becomes at least as important. In particular, exploiting context-specific test conditions may pay off in this case. Consider for instance a surveillance camera. Depending on how or where it is mounted, it always captures images under particular viewing conditions. Knowing how to cope with other conditions is no longer relevant and can be forgotten, freeing capacity for other tasks. This calls for a continual learning method that can learn what (not) to forget using unlabeled test data. We illustrate this setup in Figure 6.1.

Such adaptation and memory organization is what we also observe in biological neurosystems. Our ability to preserve what we have learned before is largely dependent on how frequently we make use of it. Skills that we practice often, appear to be unforgettable, unlike those that we have not used for a long time. Remarkably, this flexibility and adaptation occur in the absence of any form of supervision. According to Hebbian theory [55], the process at the basis of this phenomenon is the strengthening of synapses connecting neurons that fire synchronously, compared to those connecting neurons with unrelated firing behavior.

In this work, we propose a new method for continual learning, coined *Memory Aware Synapses*, or `MAS` for short, inspired by the model of Hebbian learning in biological systems. Unlike previous works, *our method can learn what parts of the model are important using unlabelled data*. This allows for adaptation to specific test conditions

and continuous updating of importance weights. This is achieved by estimating importance weights for the network parameters without relying on the loss (as in EWC [69] and SI [168]), but by looking at the sensitivity of the output function instead. This way, our method not only avoids the need for labeled data, but importantly it also avoids complications due to the loss being in a local minimum, resulting in gradients being close to zero. This makes our method not only more versatile, but also simpler, and, as it turns out, more effective in learning what not to forget, compared to other regularization based continual learning approaches.

Contributions of the work described in this chapter are threefold: *First*, we propose a new continual learning method, *Memory Aware Synapses* (MAS). It estimates importance weights for all the network parameters in an unsupervised and online manner, allowing adaptation to unlabeled data, e.g. in the actual test environment. *Second*, we show how a local variant of MAS is linked to the Hebbian learning scheme. *Third*, we achieve better performance than state-of-the-art at the time of development (2018), both when using the standard incremental learning setup and when adapting to specific test conditions, both for object recognition and for predicting <subject, predicate, object> triplets, where an embedding is used instead of a softmax output.

In the following we discuss closely related work in Section 6.2 and give some background information in Section 6.3. Section 6.4 describes our method and its connection with Hebbian learning. Experimental results are given in Section 6.5 and Section 6.6 concludes the chapter.

6.2 Related Work

The work presented in this chapter doesn't store any explicit knowledge from the previous tasks but assumes that the acquired knowledge is embedded in the model parameters. When learning a new task, the training objective is regularized to not erase important parts of previous knowledge. Our work belongs to the regularization based family (see Chapter 3, Section 3.2). Unlike the data-focused methods [85, 120, 140] that rely on distilling the knowledge from the previous task model, the prior focused methods, including ours, estimate a prior distribution over the model parameters with a mean equal to the optimal model parameters learned so far and a variance proportional to the inverse of the parameters importance weights. Most similar to our work are [69, 168]. Like them, we estimate an importance weight for each model parameter and add a regularizer when training a new task that penalizes any changes to important parameters. The difference lies in the way the importance weights are computed. In the *Elastic Weight Consolidation* work [69], this is done based on an approximation of the diagonal of the Fisher information matrix. In the *Synaptic Intelligence* work [168],

importance weights are computed during training in an online manner. To this end, they record how much the loss would change due to a change in a specific parameter and accumulate this information over the training trajectory. However, also this method has some drawbacks: 1) Relying on the weight changes in a batch gradient descent might overestimate the importance of the weights, as noted by the authors. 2) When starting from a pretrained network, as in most practical computer vision applications, some weights might be used without big changes. As a result, their importance will be underestimated. 3) The computation of the importance is done during training and fixed later. In contrast, we believe the importance of the weights should be able to adapt to the test data which the system is applied to. In contrast to the above two methods, we propose to look at the sensitivity of the learned function, rather than the loss. This simplifies the setup considerably since, unlike the loss, the learned function is not in a local minimum, so complications with gradients being close to zero are avoided.

In this work, we propose a regularization-based method that computes the importance of the network parameters not only in an online manner but also adaptive to the data that the network is tested on in an unsupervised manner. While previous works [116, 129] adapt the learning system at prediction time in a transductive setting, our goal here is to build a continual system that can adapt the importance of the weights to what the system needs to remember. Our method enjoys more desired characteristics of continual learning (see Chapter 1, Section 1.1.1), namely, constant memory, problem agnostic, adaptive, and graceful forgetting while at the same time achieving competitive performance.

6.3 Background

Notations. As in the previous chapter, we train a single, shared neural network over a sequence of tasks. The parameters $\{\theta_{ij}\}$ of the model are the weights of the connections between pairs of neurons $\mathcal{N}_i$ and $\mathcal{N}_j$ in two consecutive layers[1]. As in other prior-focused methods, our goal is then to compute an importance value Ω_{ij} for each parameter θ_{ij}, indicating its importance with respect to the previous tasks. Similarly to the settings followed in the previous chapters, we receive a sequence of tasks $\{T_t\}$ to be learned, each with its training data $D_t = (X_t, Y_t)$, with $X_t = \{x_n\}_{n=1}^{N_t}$ the input data and $Y_t = \{y_n\}_{n=1}^{N_t}$ the corresponding ground truth output data (labels). Note that we drop the task superscript from the individual samples to avoid cluttering the notations. Each task loss $\ell(f(X_t), Y_t)$, will be combined with an extra loss term to avoid forgetting. When the training procedure converges to a local minimum, the model has learned an approximation f of the true function $\bar{f}$. f maps new input samples X to the corresponding outputs $\hat{Y_1}, ..., \hat{Y_\mathcal{T}}$ for tasks $T_1 ... T_\mathcal{T}$ learned so far.

[1] In convolutional layers, parameters are shared by multiple pairs of neurons. For the sake of clarity, yet without loss of generality, we focus here on fully connected layers.

6.4 Our Approach

In the following, we introduce our approach. Like other prior-focused methods [69, 168], we estimate an importance weight for each parameter in the network. Yet in our case, these importance weights approximate the *sensitivity of the learned function* to a parameter change rather than a measure of the (inverse of) parameter uncertainty, as in [69], or the sensitivity of the loss to a parameter change, as in [168]. As it does not depend on the ground truth labels, our approach allows computing the importance using any available data (unlabeled) which in turn allows for an adaptation to user-specific settings. In a learning sequence, we start with task T_1, training the model to minimize the task loss on the training data $\ell(X_1, Y_1)$ – or simply using a pretrained model for that task.

6.4.1 Estimating Parameter Importance

After convergence, the model has learned an approximation f of the true function $\bar{f}$. f maps the input X_1 to the output $\hat{Y}_1$. This mapping f is the target we want to preserve while learning additional tasks. To this end, we measure how sensitive the function f output is to changes in the network parameters. For a given data point x_n, the output of the network is $f(x_n; \theta)$. A small perturbation $\delta = \{\delta_{ij}\}$ in the parameters $\theta = \{\theta_{ij}\}$ results in a change in the function output that can be approximated by:

$$f(x_n; \theta + \delta) - f(x_n; \theta) \approx \sum_{i,j} g_{ij}(x_n)\delta_{ij} \tag{6.1}$$

where $g_{ij}(x_n) = \frac{\partial(f(x_n;\theta))}{\partial\theta_{ij}}$ is the gradient of the learned function with respect to the parameter θ_{ij} evaluated at the data point x_n and δ_{ij} is the change in parameter θ_{ij}. Our goal is to preserve the prediction of the network (the learned function) at each observed data point and prevent changes to parameters that are important for this prediction (see Figure 6.2).

Based on Equation 6.1 and assuming a small constant change δ_{ij}, we can measure the importance of a parameter by the magnitude of the gradient g_{ij}, i.e. how much does a small perturbation to that parameter change the output of the learned function for data point x_n. We then accumulate the gradients over the given data points to obtain importance weight Ω_{ij} for parameter θ_{ij}:

$$\Omega_{ij} = \frac{1}{N}\sum_{n=1}^{N} || \, g_{ij}(x_n) \, || \tag{6.2}$$

This equation can be updated in an online fashion whenever a new data point is fed to the network. N is the total number of data points at a given phase, note that it is

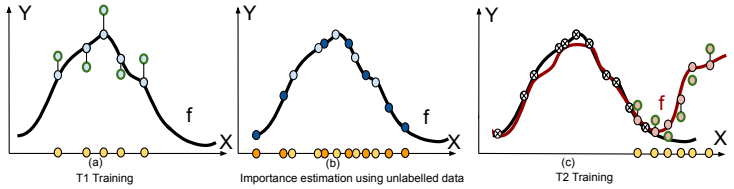


Figure 6.2: [168, 69] estimate the parameters importance based on the loss, comparing the network output (light blue) with the ground truth labels (green) using training data (in yellow) (a). In contrast, we estimate the parameters importance, after convergence, based on the sensitivity of the learned function to their changes (b). This allows using additional unlabeled data points (in orange). When learning a new task, changes to important parameters are penalized, the function is preserved over the domain densely sampled in (b), while adjusting not important parameters to ensure good performance on the new task (c).

doesn't have to be equal to N_t. Parameters with small importance weights do not affect the output much, and can, therefore, be changed to minimize the loss for subsequent tasks, while parameters with large weights should ideally be left unchanged.

When the output function f is multi-dimensional, as is the case for most neural networks, Equation 6.2 involves computing the gradients for each output, which requires as many backward passes as the dimensionality of the output. As a more efficient alternative, we propose to use the gradients of the squared ℓ_2 norm of the learned function output[2], i.e., $g_{ij}(x_n) = \frac{\partial[\ell_2^2(f(x_n;\theta))]}{\partial\theta_{ij}}$. The importance of the parameters is then measured by the sensitivity of the squared ℓ_2 norm of the function output to their changes. This way, we get one scalar value for each sample instead of a vector output. Hence, we only need to compute one backward pass and can use the resulting gradients for estimating the parameters importance. Using our method, for regions in the input space that are sampled densely, the function will be preserved and catastrophic forgetting is avoided. However, parameters not affecting those regions will be given low importance weights, and can be used to optimize the function for other tasks, affecting the function over other regions of the input space.

6.4.2 Learning a New Task

When a new task $T_{\mathcal{T}}$ needs to be learned, we have in addition to the new task loss $\ell(f(x;\theta), y)$, a regularizer that penalizes changes to parameters that are deemed important for previous tasks. As such, the training procedure of the new task minimizes the following objective function:

[2]We square the ℓ_2 norm as it simplifies the math and the link with the Hebbian method, see section 6.4.3.

$$J(\theta) = \frac{1}{N_{\mathcal{T}}} \sum_{n=1}^{N_{\mathcal{T}}} \ell(f(x_n;\theta), y_n) + \lambda \sum_{i,j} \Omega_{ij} (\theta_{ij} - \theta_{ij}^{\mathcal{T}-1})^2 \tag{6.3}$$

with $N_{\mathcal{T}}$ the number of samples of the current task, λ a hyperparameter for the regularizer and $\theta_{ij}^{\mathcal{T}-1}$ the "old" network parameters (as determined by the optimization for the previous task in the sequence, $T_{\mathcal{T}-1}$), θ is initialized with $\theta^{\mathcal{T}-1}$. Hence, we allow the new task to change parameters that are not important for the previous task (low Ω_{ij}). The important parameters (high Ω_{ij}) can also be reused, via model sharing, but with a penalty when changing them.

Finally, the importance matrix Ω is to be updated after training a new task, by accumulating over the previously computed Ω. Since we don't use the loss function, Ω can be computed on any available data considered most representative for test conditions, be it on the last training epoch, during the validation phase or at test time. In the experimental section 6.5, we show how this allows our method to adapt and specialize to any set, be it from the training or from the test.

6.4.3 Connection to Hebbian Learning

In this section, we propose a local version of our method, by applying it to a single layer of the network rather than to the network as a whole. Next, we show an interesting connection between this local version and Hebbian learning [55].

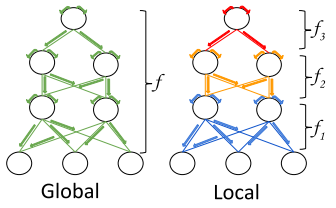


Figure 6.3: Gradients flow for computing the importance weight. Local considers the gradients of each layer independently.

A local version of our method. Instead of considering the function f that is learned by the network as a whole, we decompose it in a sequence of functions f_l each corresponding to one layer of the network, i.e., $f(x) = f_L(f_{L-1}(...(f_1(x))))$, with L the total number of layers. By locally preserving the output of each layer given its input, we can preserve the global function f. This is further illustrated in Figure 6.3. Note how "local" and "global" in this context relate to the number of layers over which the gradients are computed.

We use $h_{i,n}^l$ to denote the activation of neuron $\mathcal{N}_i$ at layer l for a given input x_n. Note that, we explicitly add a superscript l referring to the layer and for clarity we omit the sample index n when it is not needed. Hence $H_l = \{h_j^l\}$ is the output of layer l and $H_{l-1} = \{h_i^{l-1}\}$ is its input. H_l is a vector with elements $\{h_j^l\}$. θ_{ij} is then a network parameter representing the connection between neuron $\mathcal{N}_i$ in layer $l-1$ and neuron $\mathcal{N}_j$

in layer l. Analogous to the procedure followed previously, we consider the squared ℓ_2 norm of each layer after the activation function. An infinitesimal change $\delta_l = \{\delta_{ij}\}$ in the parameters $\theta_l = \{\theta_{ij}\}$ of layer l results in a change to the squared ℓ_2 norm of the local function f_l for a given input to that layer $H_{l-1} = \{h_i^{l-1}\} = f_{l-1}(...(f_1(x)))$ given by:

$$\ell_2^2(f_l(H_{l-1};\theta_l+\delta_l)) - \ell_2^2(f_l(H_{l-1};\theta_l)) \approx \sum_{i,j} g_{ij}(x)\delta_{ij} \tag{6.4}$$

where $g_{ij}(x) = \frac{\partial[\ell_2^2(f_l(H_{l-1};\theta_l))]}{\partial\theta_{ij}}$.

In the case of a ReLU activation function and considering a fully connected layer with $I * J$ parameters, i.e. $h_j^l = ReLU(out_j^l)$ with $out_j^l = \sum_{o=1}^{I} \theta_{oj} * h_o^{l-1}$, we can write

$$g_{ij}(x) = \frac{\partial[\ell_2^2(f_l(H_{l-1};\theta_l))]}{\partial\theta_{ij}} = \frac{\partial[\sum_{k=1}^{J}(h_k^l)^2]}{\partial\theta_{ij}} = \sum_{k=1}^{J} \frac{\partial((h_k^l)^2)}{\partial\theta_{ij}} \tag{6.5}$$

Since $\frac{\partial((h_k^l)^2)}{\partial\theta_{ij}} = 0$ when $k \neq j$, we get

$$g_{ij}(x) = \frac{\partial((h_j^l)^2)}{\partial\theta_{ij}} = 2 * h_j^l * \frac{\partial(h_j^l)}{\partial\theta_{ij}} = 2 * h_j^l * \frac{\partial(ReLU(out_j^l))}{\partial\theta_{ij}} \tag{6.6}$$

ReLU is non smooth since it is not differentiable at 0. We show the subgradients in the two cases (also using the fact that $out_j^l = \sum_{o=1}^{I} \theta_{oj} * h_o^{l-1}$):

1. if $out_j^l > 0$, $ReLU(out_j^l) = out_j^l$ and $(ReLU)'$ =1 :

$$\frac{\partial(ReLU(out_j^l))}{\partial\theta_{ij}} = \frac{\partial(ReLU(out_j^l))}{\partial out_j^l}\frac{\partial(out_j^l)}{\partial\theta_{ij}} = \frac{\partial(out_j^l)}{\partial\theta_{ij}}$$

$$= \frac{\partial(\sum_{o=1}^{I}\theta_{oj} * h_o^{l-1})}{\partial\theta_{ij}} = \frac{\partial(\theta_{ij} * h_i^{l-1})}{\partial\theta_{ij}} = h_i^{l-1}$$

$$\Rightarrow g_{ij}(x) = 2 * h_j^l * h_i^{l-1} \tag{6.7}$$

2. in the other case, $ReLU(out_j^l) = 0$ and $(ReLU)' = 0$, so

$$g_{ij}(x) = 0 \tag{6.8}$$

At the same time,

$$ReLU(out_j^l) = 0 \Rightarrow h_j^l = 0 \text{ and } 2 * h_j^l * h_i^{l-1} = 0 \tag{6.9}$$

Hence

$$g_{ij}(x) = 2 * h_j^l * h_i^{l-1} = 0 \tag{6.10}$$

Based on equations 6.7 and 6.10, we have shown

$$g_{ij}(x) = 2 * h_j^l * h_i^{l-1} \tag{6.11}$$

Again we consider the accumulation of the gradients evaluated at different data points $\{x_n\}$ as a measure for the importance of the parameter θ_{ij} in a layer l:

$$\Omega_{ij} = \frac{1}{N} \sum_{n=1}^{N} g_{ij}(x_n) = 2 * \frac{1}{N} \sum_{n=1}^{N} h_{j,n}^l * h_{i,n}^{l-1} \tag{6.12}$$

Link with Hebbian theory. In neuroscience, Hebbian learning theory [55] provides an explanation for the phenomenon of synaptic plasticity. It postulates that "cells that fire together, wire together": the synapses (connections) between neurons that fire synchronously for a given input are strengthened over time to maintain and possibly improve the corresponding outputs. Here we reconsider this theory from the perspective of an artificial neural network after it has been trained successfully with backpropagation. Following Hebb's rule, parameters connecting neurons that often fire together (high activations for both, i.e. highly correlated outputs) are more important for the given task than those that fire asynchronously or with low activations. As such, the importance weight Ω_{ij} for the parameter θ_{ij} in a layer l can be measured purely locally in terms of the correlation between the neurons activations, i.e.

$$\Omega_{ij} = \frac{1}{N} \sum_{n=1}^{N} h_{j,n}^l * h_{i,n}^{l-1} \tag{6.13}$$

The similarity with equation 6.12 is striking. We can conclude that applying Hebb's rule to measure the importance of the parameters in a neural network can be seen as a local variant of our method that considers only one layer at a time instead of the global function learned by the network. Since only the relative importance weights really matter, the scale factor 2 can be ignored.

6.4.4 Discussion

Our global and local methods both have the advantage of computing the importance of the parameters on any given data point without the need to access the labels or the condition of being computed while training the model. The global version needs to compute the gradients of the output function while the local variant (Hebbian based) can be computed locally by multiplying the input with the output of the connecting neurons. *Our proposed method (both the local and global version) resembles an implicit memory included for each parameter of the network. We, therefore, refer to it*

as Memory Aware Synapses. It keeps updating its value based on the activations of the network when applied to new data points. It can adapt and specialize to a given subset of data points rather than preserving every functionality in the network. Further, the method can be added after the network is trained. It can be applied on top of any pretrained network and compute the importance on any set of data without the need to have the labels. This is an important criterion that differentiates our work from methods that rely on the loss function to compute the importance of the parameters.

6.5 Experiments

We start by comparing our method to different existing continual learning methods in the standard incremental learning setup of object recognition tasks. Next, we move to the more challenging problem of continual learning of <subject, predicate, object> triplets in an embedding space, Section 6.5.2. We further analyze the behavior and some design choices of our method, Section 6.5.3.

6.5.1 Object Recognition

We follow the standard setup commonly used in computer vision to evaluate continual learning methods [83, 85, 90]. It consists of a sequence of supervised classification tasks each from a particular dataset. Note that this assumes having different classification layers for each task (different "heads") that remain unshared. Moreover, an oracle is used at test time to decide on the task (i.e., which classification layer to use).

Compared Methods.

- *Finetuning* (`Finetune`). After learning the first task and when receiving a new task to learn, the parameters of the network are finetuned on the new task data. This baseline is expected to suffer from forgetting the old tasks while being advantageous for the new task.

- *Learning without Forgetting [85]* (`LwF`). A data-focused regularization based method. As explained in previous chapters (4,5) knowledge distillation is deployed to mitigate forgetting. The method relies on first training the new task head while freezing the shared parameters as a warmup phase and then training all the parameters until convergence.

- *Encoder Based Lifelong Learning [120]* (`EBLL`). Our work detailed in Chapter 5 in which we learn a shallow encoder of the features of each task. A penalty on the changes to the encoded features accompanied with the distillation loss is applied to reduce the forgetting of the previous tasks.

- *Incremental Moment Matching [83]* (IMM). A new task is learned with an ℓ_2 penalty equally applied to the changes to all shared parameters. At the end of the sequence, the obtained models are merged through a first or second moment matching. In our experiments, mean IMM gives better results on the two tasks experiments while mode IMM wins on the longer sequence. Thus, we report the best alternative in each experiment.

- *Elastic Weight Consolidation [69]* (EWC). It is the first work introducing the prior-focused approach of the regularization based family using as importance measure the diagonal of the Fisher information matrix. EWC uses individual penalty for each previous task, however, to make it computationally feasible we apply a single penalty as pointed out by [62]. Hence, we use a running sum of the Fishers in the 8 tasks sequence.

- *Synaptic Intelligence [168]* (SI). This comes closest to our approach as it estimates the importance weights in an online manner while training for a new task. Similar to EWC and our method, changes to parameters important for previous tasks are penalized during training of later tasks.

- *Memory Aware Synapses* (MAS). Unless stated otherwise, we use the global version of our method and with the importance weights estimated only on training data. We use a regularization parameter λ of 1. Note that no tuning of λ was performed as we assume no access to previous task data.

Experimental setup. We use the AlexNet [76] architecture pretrained on ImageNet [132] from [74][3]. All the training of the different tasks has been done with stochastic gradient descent for 100 epochs and a batch size of 200 using the same learning rate as in the previous chapters. Performance is measured in terms of classification accuracy.

Two tasks experiments. We first consider sequences of two tasks based on three datasets: *Scenes* [117] for indoor scene classification, *Birds* [157] for fine-grained bird classification, and *Flowers* [111] for fine-grained flower classification. We consider: Scene$\rightarrow$ Birds, Birds$\rightarrow$ Scenes, Flowers$\rightarrow$Scenes and Flowers$\rightarrow$ Birds. We didn't consider ImageNet as a task in the sequence as this would require retraining the network from scratch to get the importance weights for SI.

As shown in Table 6.1, Finetune clearly suffers from catastrophic forgetting with a drop in performance from 8% to 13%. All the considered methods manage to reduce the forgetting over fine-tuning significantly while having performance close to fine-tuning on the new task. On average, our method achieves the lowest forgetting rates (around 1%) while performance on the new task is almost similar ($0-3\%$ lower).

[3] We use the pretrained model available in Pytorch. Note that it differs slightly from other implementations used e.g. in [85].

Method	Birds → Scenes		Scenes → Birds		Flowers → Birds		Flowers → Scenes	
Finetune	45.20 (8.0)	**57.8**	49.7 (9.3)	**52.8**	64.87 (13.2)	**53.8**	70.17 (7.9)	57.31
LwF [85]	51.65 (2.0)	55.59	55.89 (3.1)	49.46	73.97 (4.1)	53.64	76.20 (1.9)	58.05
EBLL [120]	52.79 (0.8)	55.67	56.34 (2.7)	49.41	75.45 (2.6)	50.51	76.20 (1.9)	**58.35**
IMM [83]	51.51 (2.1)	52.62	54.76 (4.2)	52.20	75.68 (2.4)	48.32	76.28 (1.8)	55.64
EWC [69]	52.19 (1.4)	55.74	**58.28 (0.8)**	49.65	76.46 (1.6)	50.7	77.0 (1.1)	57.53
SI [168]	52.64 (1.0)	55.89	57.46 (1.5)	49.70	75.19 (2.9)	51.20	76.61 (1.5)	57.53
MAS (ours)	**53.24 (0.4)**	55.0	57.61 (1.4)	49.62	**77.33 (0.7)**	50.39	**77.24 (0.8)**	57.38

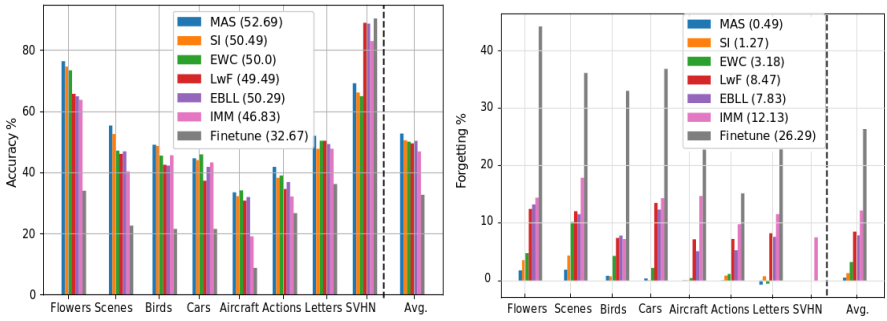
Table 6.1: Classification accuracy (%), forgetting on the first task (%) for various sequences of 2 tasks using the object recognition setup.

Method	Ω computed. on	Birds → Scenes		Scenes → Birds		Flowers → Birds		Flowers → Scenes	
MAS	Train	53.24 (0.4)	55.0	57.61 (1.4)	49.62	77.33 (0.7)	50.39	77.24 (0.8)	57.38
MAS	Test	53.43 (0.2)	55.07	57.31 (1.7)	49.01	77.62 (0.5)	50.29	77.45 (0.6)	57.45
MAS	Train + Test	53.29 (0.3)	56.04	57.83 (1.2)	49.56	77.52 (0.6)	49.70	77.54 (0.5)	57.39
l-MAS	Train	51.36 (2.3)	55.67	57.61 (1.4)	49.86	73.96 (4.1)	50.5	76.20 (1.9)	56.68
l-MAS	Test	51.62 (2.0)	53.95	55.74 (3.3)	50.43	74.48 (3.6)	50.32	76.56 (1.5)	57.83
l-MAS	Train + Test	52.15 (1.5)	54.40	56.79 (2.2)	48.92	73.73 (4.3)	50.5	76.41 (1.7)	57.91

Table 6.2: Classification accuracies (%) for the object recognition setup - comparison between using Train and Test data (unlabeled) to compute the parameter importance Ω.

Local vs. global MAS on training/test data. Next we analyze the performance of our method when preserving the global function learned by the network after each task (MAS) and its local Hebbian-inspired variant described in section 6.4.3 (l-MAS). We evaluate our methods, MAS and l-MAS, when using unlabeled test data and/or labeled training data. Table 6.2 shows for both l-MAS and MAS, the preservation of the previous task and the performance on the current task are quite similar given the different used sets of data. This illustrates our methods ability to estimate the parameters importance of a given task given any set of points, without the need for labeled data. Further, computing the gradients locally at each layer for l-MAS allows for faster computations but less accurate estimations. As such, l-MAS shows an average forgetting of 3% compared to 1% by MAS.

ℓ_2^2 vs. vector output. We explained in section 6.4 that considering the gradients of the learned function to estimate the parameters importance would require as many backward passes as the length of the output vector. To avoid this complexity, we suggest using the square of the ℓ_2 norm of the function to get a scalar output. We run two experiments, Flowers→ Scenes and Flowers→ Birds once with computing the gradients with respect to the vector output and once with respect to the ℓ_2^2 norm. We observe no significant difference on forgetting over 3 random trials where we get a mean, over 6 numbers, of $0.51\% \pm 0.18$ for the forgetting on the first task in the vector output case compared to $0.50\% \pm 0.19$ for the ℓ_2^2 norm case. No significant difference is observed on the second task either. As such, we can conclude that using ℓ_2^2 is m times faster (where m is the length of the output vector) without loss in performance.



(a) (b)

Figure 6.4: (a) Performance on each task, in accuracy, at the end of the 8 tasks object recognition sequence. (b) Forgetting on each task measured at the end of the sequence, lower is better.

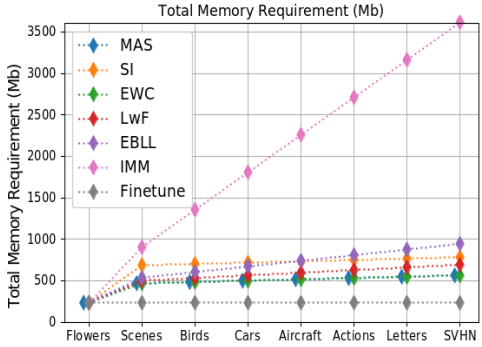


Figure 6.5: Overall memory requirement for each method at each step of the sequence.

Longer Sequence While the two tasks setup gives a detailed look at the average expected forgetting when learning a new task, it remains easy. Thus, we next consider a sequence of 8 tasks. To do so, we add five more datasets: `Cars` [73] for fine-grained car classification; `Aircraft` [94] for fine-grained aircraft classification; `Actions`, for human action classification [36]; `Letters` [31] for character recognition in natural images; and `SVHN` dataset [108] for digit recognition. We have also used those datasets in the previous chapters. We run the different methods on the following sequence:

Flowers→Scenes→Birds→Cars→Aircraft→Actions→Letters→SVHN.

While Figure 6.4a shows the performance on each task at the end of the sequence, 6.4b shows the observed forgetting on each task at the end of the sequence (relative to the performance right after training that task). The differences between the compared

methods become more outspoken. Finetune suffers from a severe forgetting on the previous tasks while being advantageous for the last task, as expected. LwF [85] suffers from a buildup of errors when facing a long sequence while EBLL [120] reduces this effect. IMM [83] merges the models at the end of the sequence and the drop in performance differs between tasks. More importantly, the method performance on the last task is highly affected by the moment matching. SI [168] followed by EWC [69] has the least forgetting among our methods competitors. MAS, our method, shows a minimal or no forgetting on the different tasks in the sequence with an average forgetting of 0.49%. It is worth noting that our method's absolute performance on average including the last task is 2% better than SI which indicates our method ability to accurately estimate the importance weights and to allow the new tasks to adjust accordingly. Apart from evaluating forgetting, we analyze the memory requirements of each of the compared methods. Figure 6.5 illustrates the memory usage of each method at each learning step in the sequence. After Finetune that doesn't treat forgetting, our method has the lowest memory consumption. Note that IMM grows linearly in storage, but at inference time it only uses the obtained model.

6.5.2 Fact Learning

Next, we move to a more challenging setup where all the layers of the network are shared, including the last layer. Instead of learning a classifier, we learn an embedding space. For this setup, we pick the problem of Fact Learning from natural images [35]. For example, a fact could be "person eating pizza". We design different experimental settings to show the ability of our method to learn what (not) to forget.

Experimental setup. We use the 6DS mid scale dataset presented in [35]. It consists of $28,624$ images, divided equally in training and test samples belonging to 186 unique facts. Facts are structured into 3 units: Subject (S), Object (O) and Predicate (P). We use a CNN model based on the VGG-16 architecture [143] pretrained on ImageNet. The last fully connected layer forks in three final layers enabling the model to have three separated and structured outputs for Subject, Predicate and Object as in [35]. The loss minimizes the pairwise distance between the visual and the language embedding. For the language embedding, the Word2vec [103] representation of the fact units is used. To study fact learning from a continual perspective, we divided randomly the dataset into tasks belonging to different groups of facts. SGD optimizer is used with a mini-batch of size 35 for 300 epochs and we use a $\lambda = 5$ for our method. For evaluation, we report the fact to image retrieval scenario. We follow the evaluation protocol proposed in [35] and report the mean average precision (MAP). For each task, we consider retrieving the images belonging to facts from this task only. We also report the mean average precision on the whole dataset which differs from the average of the performance achieved on each task. We focus on the comparison between MAS, our

Method	Split	Method evaluated on T_1	T_2	T_3	T_4	all
Finetune	1	0.19	0.19	0.28	**0.71**	0.18
SI[168]	1	0.36	0.32	0.38	0.68	0.25
MAS (ours)	1	**0.42**	**0.37**	**0.41**	0.65	**0.29**
Finetune	2	0.20	0.27	0.18	0.66	0.18
SI[168]	2	0.37	0.39	0.38	0.46	0.24
MAS (ours)	2	**0.42**	**0.42**	**0.46**	**0.65**	**0.28**
Finetune	3	0.21	0.25	0.24	0.46	0.14
SI [168]	3	**0.30**	0.31	0.36	0.61	0.24
MAS (ours)	3	**0.30**	**0.36**	**0.38**	**0.66**	**0.27**

Table 6.3: MAP for fact learning on the 4 tasks random split, from the 6DS dataset, at the end of the sequence.

method, and SI [168], the best performing method among the different competitors as shown in Figure 6.4a.

Four tasks experiments We consider a sequence of 4 tasks obtained from splitting randomly the facts of the same dataset into 4 groups. Table 6.3 presents the achieved performance on each set of the 4 tasks at the end of the learned sequence based on 3 different random splits. Similar to previous experiments, Finetune is only advantageous on the last task while drastically suffering on the previous tasks. However, here, our method differentiates itself clearly, showing 6 better MAP on the first two tasks compared to SI. Overall, MAS achieves a MAP of 0.29 compared to 0.25 by SI and only 0.18 by Finetune. When MAS importance weights are computed on both training and test data, a further improvement is achieved with 0.30 overall performance. This highlights our method ability to benefit from extra unlabeled data to further enhance the importance estimation.

6.5.3 Behavior Analysis

Sensitivity to the hyper parameter. Our method needs one extra hyper parameter, λ, that weights the penalty on the parameters changes as shown in Eq 6.3. λ is a trade-off between the allowed forgetting and the new task loss. We set λ to the largest value that allows an acceptable performance on the new task. for SI[168] and EWC[69] we had to vary λ. Figure 6.6 shows the effect of λ on the Avg. performance and the Avg. forgetting in a sequence of 5 permuted MNIST tasks with a 2 layer perceptron (512 units). We see the sensitivity around $\lambda = 1$ is very low with low forgetting. Note that a better trade-off could be achieved by setting λ to the smallest value that allows an acceptable low forgetting on the previous tasks. However, when moving to a new task we assume no access to the previous tasks data and hence such a strategy can't be followed.

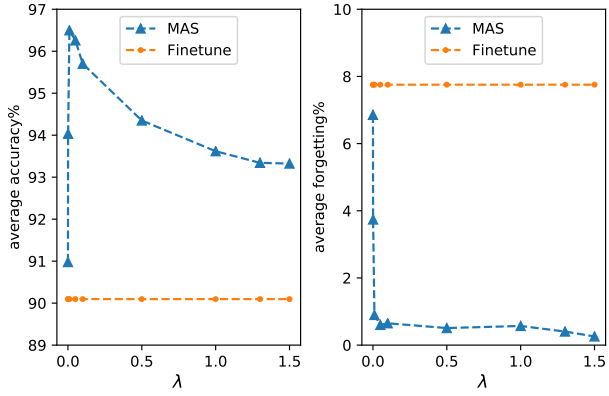


Figure 6.6: Avg. performance, left, and Avg. forgetting, right, on permuted MNIST sequence.

Object recognition adaptation test. As we have previously explained, `MAS` has the ability to adapt the importance weights to a specific subset that has been encountered at test time in an unsupervised and online manner. To test this claim, we have selected one class from the `Flowers` dataset, `Krishna Kamal` flower. We learn the 8 tasks sequence as above while assuming `Krishna Kamal` as the only encountered class. Hence, importance weights are computed on that subset only. At the end of the sequence, we observe a minimal forgetting on that subset of 2% compared to 8% forgetting on the `Flowers` dataset as a whole. We also observe higher accuracies on later tasks as only changes to important parameters for that class are penalized, leaving more free capacity for remaining tasks (e.g. accuracy of 84% on the last task, instead of 69% without adaptation). We repeat the experiment with two other classes and obtain similar results. This clearly indicates our method ability to adapt to user specific settings and to learn what (not) to forget.

Fact learning adaptation test. We want to test the adaptation ability of our method, here on a specific subset of facts assumed to be frequently encountered at test time. To proceed, we clustered the dataset into 4 disjoint groups of facts, representing 4 tasks, and then selected a specialized subset of T_1, namely 7 facts of person playing sports. We run our method with the importance parameters computed only over the examples from this set along the 4 tasks sequence. Figure 6.7 shows the achieved performance on this sport subset by each method at each step of the learning sequence. Joint Training (black dashed) is shown as reference. It violates the continual learning setting as it trains on all data jointly. Note that `SI` can only learn importance weights during training, and therefore cannot adapt to a particular subset. Our `MAS` (pink)

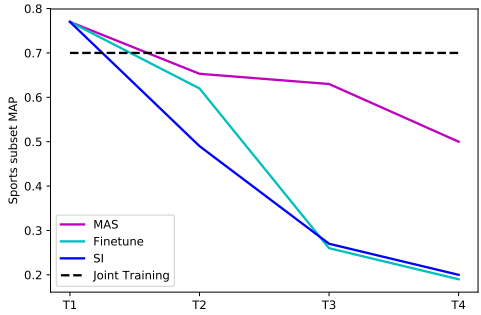
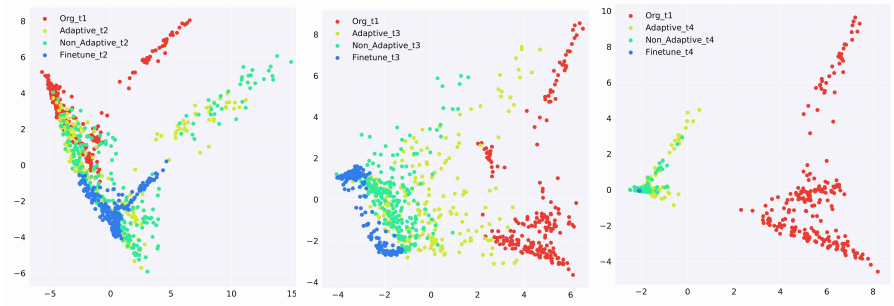


Figure 6.7: MAP on the sport subset of the 6DS dataset after each task in a 4 tasks sequence. `MAS` managed to learn that the sport subset is important to preserve and prevents significantly the forgetting on this subset.



(a) (b) (c)

Figure 6.8: Projections onto a 2D embedding, after training the second task (a), after training the third task (b) and after training the fourth task (c).

succeeds to learn that this set is important to preserve and achieves a performance of 0.50 at the end of the sequence, while the performance of `Finetune` and `SI` on this set was close to 0.20.

Visualizing the learned embedding. We have showed that our method reduces the forgetting on a specific subset of a task the most among the competitors that do not have this specialization capabilities, Fig. 6.7. Now, we want to know what happens in the learned embedding space, i.e. how the projections of the samples that belong to this subset change along the sequence compared to how they were right after training the first task. For that purpose, we extracted tSNE $2D$ projection of the learned embedding

after each task in the sequence. This was done for our method (MAS) when adapting to sport subset (Adaptive) and our method (MAS) when preserving the performance on all facts of the first task (Non Adaptive). We also show the projections of the points in the embedding learned by the finetune baseline (Finetune, where no regularizer is used). To have a point of reference, we also show the projections of the originally learned representation after the first task (Org). Figure 6.8a shows the projections from the different variants after learning the second task compared to the original projections. It can be seen that the Adaptive and Non Adaptive variants of our method try to preserve the projections from this subset. The adaptive projections are closer to the original one, if we look closely, while Finetune projections start drifting away from where they were. After the third task, as shown in figure 6.8b, the Adaptive projections are closer to the original ones than the Non Adaptive that considers this subset as part of the full task being preserved and tries to prevent forgetting them as well. Finetune started destroying the learned topology of this subset and lies further apart. However, when it comes to the fourth task, we see that it is a quite challenging and hard task. The forgetting appears more severe than before and preservation of the projections become even harder. Nevertheless, the Adaptive MAS and Non Adaptive MAS still preserve the topology of the learned projections. The Adaptive projections lie closer and look more similar to the originals than the Non Adaptive MAS. Finetune, on the other hand, forgets completely about this subset and all the samples get projected in one point where it becomes quite hard to recognize their corresponding facts.

Correlation between the parameters importance computed on different sets. We want to investigate the correlation or the difference between the importance assigned to the parameters computed on different sets.

First, we compare the estimated parameters importance (Ω) using the training data and Ω computed using the test data. For that, we used a model from the object recognition experiment, namely Birds→Scenes, the results of which are shown in Table 6.1. Figure 6.9 shows a scatter plot for the top 1000 most important parameters according to the Ω computed on the training data (blue). The X-axis represents the values from Ω computed on training data while the Y-axis represents the values from Ω computed on test data. Figure 6.10 shows a similar scatter plot for the top 1000 important parameters according to the Ω computed on the test data (red). Here, the X-axis represents the values from Ω computed on test data while the Y-axis represents the values from Ω computed on training data. A plot where the points are closely lying around a straight line indicates that the parameters from the two Ωs have similar importance values. A plot where the points are spread further from such a line and scattered among the plotted area indicates a lower correlation between the Ωs. It can be seen how similar are the importance values computed on test data to those computed on training data where they form a tight grouping of points around a straight line where the values would be identical. This demonstrates our method's ability to correctly identify the important parameters in an unsupervised manner, regardless of what set is used for that

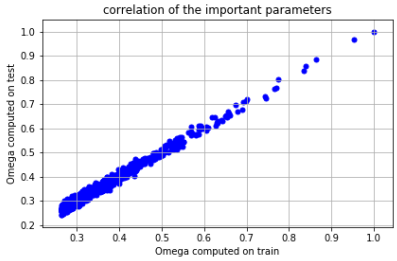


Figure 6.9: Top most important parameters from Ω computed on training data. The X-axis represents the values from Ω computed on training data while the Y-axis represents the values from Ω computed on test data. Importance shown for the last fully connected layer based on object recognition experiment Birds→Scenes

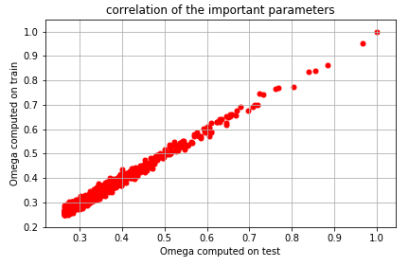


Figure 6.10: Top important parameters from Ω computed on test data. The X-axis represents the values from Ω computed on test data while the Y-axis represents the values from Ω computed on training data. Importance shown for the last fully connected layer based on object recognition experiment Birds→Scenes

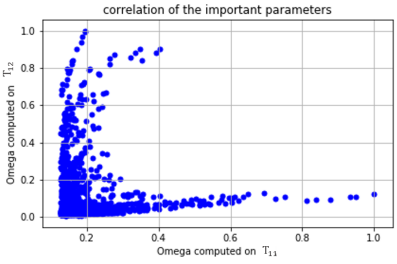


Figure 6.11: Top most important parameters from Ω computed on T_{11}. The X-axis represents the values from Ω computed on T_{11} while the Y-axis represents the values from Ω computed on T_{12}. Importance shown for the last convolutional layer based on two tasks split of the fact learning dataset, 6DS.

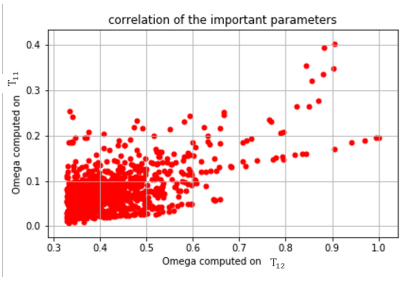


Figure 6.12: Top most important parameters from Ω computed on T_{12}. The X-axis represents the values from Ω computed on T_{12} while the Y-axis represents the values from Ω computed on T_{11}. Importance shown for the last convolutional layer based on two tasks split of the fact learning dataset, 6DS.

purpose as long as it covers the different classes or concepts of the task at hand.

How about using different subsets that cover a partial set of classes or concepts from

a task? We have shown that computing the importance on one subset results in a better preservation of performance compared with the other subset that was not used for computing the importance. This suggests that the importance of the parameters differs while using different subsets. To further investigate this claim, we split the Fact learning dataset, 6DS, into two groups of randomly selected facts resulting in two tasks. We then split the test set of the first task data into two random subsets of facts, T_{11} and T_{12}. After learning the task T_1, the importance of the parameters is computed using one subset only (T_{11} or T_{12}). We plotted the values of Ω for the 1000 top most important parameters estimated on the T_{11} (in blue) subset of the training data from the first task T_1 along with the same parameters but with their importance computed using the other subset T_{12}, Figures 6.11 and 6.12.

This suggests that the method identifies the important parameters needed for each subset and when those parameters are shared the parameters importance is correlated between the two subsets while when those are different, different parameters receive different importance values based on the used subset.

6.6 Summary

In this chapter, we argued that given a limited model capacity and unlimited evolving tasks, it is not possible to preserve all the previous knowledge. Instead, agents should learn what (not) to forget. Forgetting should relate to the rate at which a specific piece of knowledge is used. This is similar to how biological systems are learning. In the absence of error signals, synapses connecting biological neurons strengthen or weaken based on the concurrence of the connected neurons activations. Inspired by the synaptic plasticity, we proposed a method that is able to learn the importance of network parameters from the input data that the system is active on, in an unsupervised manner. We showed that a local variant of our method can be seen as an application of Hebb's rule in learning the importance of parameters. We first tested our method on a sequence of object recognition problems in a traditional incremental task learning setting. We then moved to a more challenging test case where we learn facts from images in a continuous manner. We showed i) the ability of our method to better learn the importance of the parameters using training data, test data or both; ii) at time of development (2018) state-of-the-art performance on all the designed experiments and iii) the ability of our method to adapt the importance of the parameters towards a frequent set of data. We believe that this is a step forward in developing systems that can always learn and adapt in a flexible manner.

Chapter 7

Sparsity in Continual Learning

In this chapter, we study the role of sparsity and representation decorrelation in the context of continual learning. Given a scenario with fixed model capacity, we postulate that the learning process should not be selfish, i.e. it should account for future tasks to be added and thus leave enough capacity for them. To achieve *Selfless Continual Learning* we study different regularization strategies and activation functions. We find that imposing sparsity at the level of the representation (i.e. neuron activations) is more beneficial for continual learning than encouraging parameter sparsity.

In particular, we propose a novel regularizer, that encourages representation sparsity by means of neural inhibition. It results in few active neurons which in turn leaves more free neurons to be utilized by upcoming tasks. As neural inhibition over an entire layer can be too drastic, especially for complex tasks requiring strong representations, our regularizer only inhibits other neurons in a local neighbourhood, inspired by lateral inhibition processes in the brain. We combine our novel regularizer with MAS (see Chapter 6), our method that penalizes changes to important previously learned parts of the network. We show that our new regularizer leads to increased sparsity which translates in consistent performance improvement on diverse datasets.

This work was also a collaboration with Marcus Rohrbach from FAIR. It was published as an article in ICLR 2019 [10].

7.1 Introduction

The key challenge of continual learning is how to avoid catastrophic interference with the tasks learned previously [42]. Similar to the previous chapters (5, 6), we are

interested in learning a sequence of tasks *without access to any previous or future task data* and *restricted to a fixed model capacity*, as also studied in [69, 38, 96, 136]. This scenario not only has many practical benefits, including privacy and scalability, but also resembles more closely how the mammalian brain learns tasks over time.

The mammalian brain is composed of billions of neurons. Yet at any given time, information is represented by only a few active neurons [84, 1]. In neural biology, *lateral inhibition* describes the process where an activated neuron reduces the activity of its weaker neighbors. This creates a powerful decorrelated and compact representation with minimum interference between different input patterns in the brain [165, 89]. This is in stark contrast with artificial neural networks, which typically learn dense representations that are highly entangled [17]. Such an entangled representation is quite sensitive to changes in the input patterns, in that it responds differently to input patterns with only small variations. [42] suggests that an overlapped representation plays a crucial role in catastrophic forgetting and reducing this overlap would result in reduced interference. [28] shows that when the amount of overfitting in a neural network is reduced, the representation correlation is also reduced. As such, learning a disentangled representation is more powerful and less vulnerable to catastrophic interference. However, if the learned disentangled representation at a given task is not sparse, only little capacity is left for learning new tasks. This would in turn result in either an underfitting to the new tasks or again a forgetting of previous tasks. In contrast, a sparse and decorrelated representation would lead to a powerful representation and at the same time enough free neurons that can be changed without interference with the neural activations learned for the previous tasks.

In general, sparsity in neural networks can be thought of either in terms of the network parameters or in terms of the representation (i.e. the activations). In this chapter we postulate, and confirm experimentally, that a sparse and decorrelated representation is preferable over parameter sparsity in a sequential learning scenario. There are two arguments for this: first, a sparse representation is less sensitive to new and different patterns (such as data from new tasks) and second, the training procedure of the new tasks can use the free neurons leading to less interference with the previous tasks, hence reducing forgetting. In contrast, when the effective parameters are spread among different neurons, changing the ineffective ones would change the function of their corresponding neurons and hence interfere with previous tasks (see also Figure 7.1). Based on these observations, we propose a new regularizer that exhibits a behavior similar to the lateral inhibition in biological neurons. The main idea of our regularizer is to penalize neurons that are active at the same time. This leads to more sparsity and a decorrelated representation. However, complex tasks may actually require multiple active neurons in a layer at the same time to learn a strong representation. Therefore, our regularizer, **Sparse coding through Local Neural Inhibition and Discounting** (`SLNID`), only penalizes neurons locally. Furthermore, we don't want inhibition to affect previously learned tasks, even if later tasks use neurons from earlier tasks. An

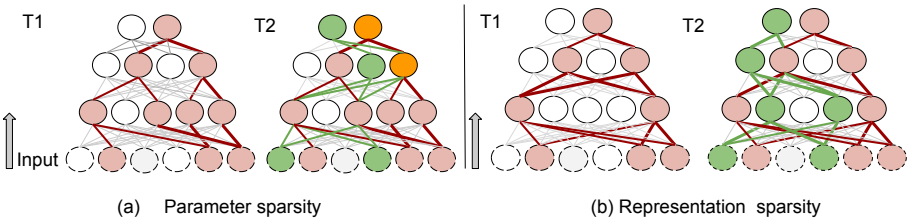


Figure 7.1: The difference between parameter sparsity (a) and representation sparsity (b) in a simple two tasks case. First layer indicates input patterns. Learning the first task utilizes parts indicated in red. Task 2 has different input patterns and uses parts shown in green. Orange indicates changed neurons activations as a result of the second task. In (a), when an example from the first task is encountered again, the activations of the first layer will not be affected by the changes, however, the second and later layer activations are changed. Such interference is largely reduced when imposing sparsity on the representation (b).

important component of SLNID is thus to discount inhibition from/to neurons which have high *neuron importance* – a new concept that we introduce in analogy to parameter importance [69, 168, 5]. When combined with a prior focused method [5, 69], our proposed regularizer leads to sparse and decorrelated representations which improve the continual learning performance.

The contribution of the work presented in this chapter is threefold. First, we direct attention to Selfless Continual Learning and study a diverse set of representation based regularizers, parameter based regularizers, as well as sparsity inducing activation functions. These have not been studied extensively in the continual learning literature before. Second, we propose a novel regularizer, SLNID, which is inspired by lateral inhibition in the brain. Third, we show that our proposed regularizer consistently outperforms alternatives on three diverse datasets (Permuted MNIST, CIFAR, Tiny ImageNet) and we compare to and outperform state-of-the-art continual learning approaches at the time of development (2018) on the 8 tasks object classification sequence (see Chapter 6, Section 6.5.1) . SLNID can be applied to different regularization based continual learning methods, and we show experiments with MAS [5] and EWC [69].

In the following, we first discuss closely related continual learning methods and different regularization criteria from a continual learning perspective (Section 7.2). We proceed by introducing Selfless Continual Learning and detailing our novel regularizer (Section 7.3). Section 7.4 describes our experimental evaluation, while Section 7.5 concludes the chapter.

7.2 Related Work

In this chapter, we focus on learning a sequence of tasks using a fixed model capacity, i.e. with a fixed architecture and fixed number of parameters. Under this setting, we consider the prior-focused branch of the regularization based family [69, 168, 24, 90] (see Chapter 3, Section 3.2).

A common drawback of all methods in this family is that learning a task could utilize a good portion of the network capacity, leaving few "free" neurons to be adapted by the new task. This in turn leads to inferior performance on the newly learned tasks or forgetting the previously learned ones, as we will show in the experiments.

While in the previous chapter we presented our method, MAS, that learns what is important to remember leaving the future tasks with more freedom to adjust the network parameters, here we study the role of sparsity and representation decorrelation in continual learning. This aspect has not received much attention in the literature yet. Recently, [136] proposed to overcome catastrophic forgetting through learned hard attention masks for each task with ℓ_1 regularization imposed on the accumulated hard attention masks. This comes closer to our approach although we study and propose a regularization scheme on the learned representation.

The concept of reducing the representation overlap was suggested before in early attempts towards overcoming catastrophic forgetting in neural networks [42]. This led to several methods with the goal of orthogonalizing the activations [39, 40, 78, 79, 144]. However, these approaches are mainly designed for specific architectures and activation functions, which makes it hard to integrate them in recent neural network structures.

The sparsification of neural networks has mostly been studied for compression. SVD decomposition can be applied to reduce the number of effective parameters [161]. However, there is no guarantee that the training procedure converges to a low rank weight matrix. Other works iterate between pruning and retraining of a neural network as a post processing step [88, 149, 2, 93]. While compressing a neural network by removing parameters leads to a sparser neural network, this does not necessarily lead to a sparser representation. Indeed, a weight vector can be highly sparse but spread among the different neurons. This reduces the effective size of a neural network, from a compression point of view, but it would not be beneficial for later tasks as most of the neurons are already occupied by the current set of tasks. In our experiments, we show the difference between using a sparse penalty on the representation versus applying it to the weights.

7.3 Selfless Continual Learning

One of the main challenges in single model incremental learning is to have capacity to learn new tasks and at the same time avoid catastrophic forgetting of previous tasks as a result of learning new tasks. In order to prevent catastrophic forgetting, as we have shown in the previous chapter, importance weight based methods like our MAS [5] or EWC [69] introduce an importance weight Ω_{ij} for each parameter θ_{ij} in the network. While each method differs in how to estimate the important parameters, changes to important parameters when learning a new task $T_{\mathcal{T}}$ are typically discouraged using an ℓ_2 penalty:

$$J(\theta) = \quad \frac{1}{N_{\mathcal{T}}} \sum_{n=1}^{N_{\mathcal{T}}} \ell(f(x_n;\theta), y_n) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta_{ij}^{\mathcal{T}-1})^2 \tag{7.1}$$

where $\theta^{\mathcal{T}-1} = \{\theta_{ij}^{\mathcal{T}-1}\}$ are the optimal parameters learned so far, i.e. before the current task. $\{x_n\}$ is the set of $N_{\mathcal{T}}$ training inputs, with $\{f(x_n;\theta)\}$ and $\{y_n\}$ the corresponding predicted and desired outputs, respectively. λ is a trade-off parameter between the new task objective ℓ and the changes on the important parameters, i.e. the amount of forgetting.

In this work, we introduce an additional regularizer R_{SCL} (SCL short for Selfless Continual Learning). It encourages sparsity in the activations $H_l = \{h_j^n\}$ for each layer l. The training of the new task minimizes the following objective function:

$$J(\theta) = \quad \frac{1}{N_{\mathcal{T}}} \sum_{n=1}^{N_{\mathcal{T}}} \ell(f(x_n;\theta), y_n) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta_{ij}^{\mathcal{T}-1})^2 + \lambda_{\text{SCL}} \sum_{l} R_{\text{SCL}}(H_l) \tag{7.2}$$

λ_{SCL} and λ are trade-off parameters that control the contribution of each term. When training the first task ($\mathcal{T} = 1$), $\Omega_{ij} = 0$.

7.3.1 Sparse Coding through Neural Inhibition

We consider a hidden layer l with activations $H_l = \{h_j^n\}$ for a set of input samples $X = \{x_n\}_{n=1}^{N_{\mathcal{T}}}$ in a task $T_{\mathcal{T}}$, and $j \in 1, .., J$ running over all J neurons in the hidden layer. Here, we describe how we obtain a sparse and decorrelated representation. In the literature, sparsity has been proposed by [46] to be combined with the rectifier activation function (ReLU) to control unbounded activations and to increase sparsity. They minimize the ℓ_1 norm of the activations (since minimizing the ℓ_0 norm is an NP hard problem). However, ℓ_1 norm imposes an equal penalty on all the active neurons leading to small activation magnitude across the network.

Learning a decorrelated representation, on the other hand, has been explored before with the goal of reducing overfitting. This is usually done by minimizing the Frobenius norm of the covariance matrix corrected by the diagonal, as in [28] or [158]. More formally, decorrelating the representation following [28] can be imposed by Equation 7.3:

$$R(H_l) = \sum_{j,k} \Big(\frac{1}{N_{\mathcal{T}}} \sum_{n} (h_j^n - \mu_j)(h_k^n - \mu_k) \Big)^2, \quad k \neq j \tag{7.3}$$

where $j, k \in 1, .., J$. Such a penalty results in a decorrelated representation but with activations that are mostly close to a non zero mean value. We merge the two objectives of sparse and decorrelated representation in the following objective:

$$R_{\texttt{SNI}}(H_l) = \frac{1}{N_{\mathcal{T}}} \sum_{j,k} \sum_{n} h_j^n h_k^n, \quad k \neq j \tag{7.4}$$

This formula differs from minimizing the Frobenius norm of the covariance matrix in two simple yet important aspects:
(1) In the case of a ReLU activation function, used in most modern architectures, a neuron is active if its output is larger than zero, and zero otherwise. By assuming a close to zero mean of the activations, $\mu_j \simeq 0 \, \forall j \in 1, .., J$, we minimize the correlation between any two active neurons.
(2) We don't use the Frobenius norm and therefor by evaluating the derivative of the presented regularizer w.r.t. the activation, we get:

$$\frac{\partial R_{\texttt{SNI}}(H_l)}{\partial h_j^n} = \frac{1}{N_{\mathcal{T}}} \sum_{k \neq j} h_k^n \tag{7.5}$$

i.e., each active neuron receives a penalty from every other active neuron that corresponds to that other neuron's activation magnitude. In other words, if a neuron fires, with a high activation value, for a given example, it will suppress firing of other neurons for that same example. Hence, this results in a decorrelated sparse representation. It is worth noting that we impose sparsity per example, allowing our method to be extended to online setting. We hypothesize that neurons firing strongly for a given example will fire for examples with similar patterns. As such imposing per example sparsity would lead to neuron activation sparsity across different examples. We empirically validate our hypothesis in the experiments Section 7.4.1.

7.3.2 Sparse Coding through Local Neural Inhibition

The loss imposed by the SNI objective will only be zero when there is at most one active neuron per example. This seems to be too harsh for complex tasks that need a richer

representation. Thus, we suggest to relax the objective by imposing a spatial weighting to the correlation penalty. In other words, an active neuron penalizes mostly its close neighbours and this effect vanishes for neurons further away. Instead of uniformly penalizing all the correlated neurons, we weight the correlation penalty between two neurons with locations k and j using a Gaussian weighting. This gives

$$R_{\texttt{SLNI}}(H_l) = \frac{1}{N_{\mathcal{T}}} \sum_{j,k} e^{-\frac{(j-k)^2}{2\sigma^2}} \sum_n h_j^n h_k^n, \quad j \neq k \tag{7.6}$$

As such, each active neuron inhibits its neighbours, introducing a locality in the network inspired by biological neurons. Note that by adding the local weighting we impose a topology on each layer of the network. While the notion of neighbouring neurons is not well established in a fully connected network, our aim is to allow few neurons to be active and not only one, thus those few activations don't have to be small to compensate for the penalty. σ^2 is a hyper parameter representing the scale at which neurons can affect each other. Note that this is somewhat more flexible than decorrelating neurons in fixed groups as used in [158]. Our regularizer inhibits locally the active neurons leading to a sparse coding through local neural inhibition.

7.3.3 Neuron Importance for Discounting Inhibition

Our regularizer is to be applied for each task in the learning sequence. In the case of tasks with completely different input patterns, the active neurons of the previous tasks will not be activated given the new tasks input patterns. However, when the new tasks are of similar or shared patterns, neurons used for previous tasks will be active. In that case, our penalty would discourage other neurons from being active and encourage the new task to adapt the already active neurons instead. This would interfere with the previous tasks and could increase forgetting which is exactly what we want to overcome. To avoid such interference, we add a weight factor taking into account the importance of the neurons with respect to the previous tasks. To estimate the importance of the neurons, we use as a measure the sensitivity of the loss at the end of the training to their changes. This is approximated by the gradients of the loss w.r.t. the neurons outputs (before the activation function) evaluated at each data point. To get an importance value, we then accumulate the absolute value of the gradients over the given data points obtaining importance weight α_j for neuron $\mathcal{N}_j$:

$$\alpha_j = \frac{1}{N_t} \sum_{n=1}^{N_t} \mid g_j(x_n) \mid \quad , \quad g_j(x_n) = \frac{\partial(\ell(f(x_n;\theta_t), y_n))}{\partial out_j^n} \tag{7.7}$$

where out_j^n is the output of neuron $\mathcal{N}_j$ for a given input example x_n, and θ^t are the parameters after learning task T_t. This is in line with the estimation of the parameters importance in EWC [69] but considering the derivation variables to be the neurons

outputs instead of the parameters.
Instead of relying on the gradient of the loss, we can also use the gradient of the learned function, i.e. the output layer, as we have proposed in Chapter 6 for estimating the parameters importance. During the early phases of this work, we experimented with both and observed a similar behaviour. For sake of consistency and computational efficiency we utilize the gradient of the function when using our MAS [5] as continual learning method and the gradient of the loss when experimenting with EWC [69]. Then, we can weight our regularizer as follows:

$$R_{\texttt{SLNID}}(H_l) = \frac{1}{N_{\mathcal{T}}} \sum_{j,k} e^{-(\alpha_j + \alpha_k)} e^{-\frac{(j-k)^2}{2\sigma^2}} \sum_n h_j^n h_k^n, \quad k \neq j \tag{7.8}$$

which can be read as: if an important neuron for a previous task is active given an input pattern from the current task, it will not suppress the other neurons from being active neither be affected by other active neurons. For all other active neurons, local inhibition is deployed. The final objective for training is given in Equation 7.2, setting $R_{SCL} := R_{\texttt{SLNID}}$ and $\lambda_{\text{SCL}} := \lambda_{\texttt{SLNID}}$. We refer to our full method as Sparse coding through Local Neural Inhibition and Discounting (SLNID).

7.4 Experiments

In this section we study the role of standard regularization techniques with a focus on sparsity and decorrelation of the representation in a task incremental scenario. We first compare different activation functions and regularization techniques, including our proposed SLNID, on permuted MNIST (Section 7.4.1). Then, we compare the top competing techniques and our proposed method in the case of incrementally learning Cifar-100 classes and Tiny ImageNet classes (Section 7.4.3). Our SLNID regularizer can be integrated in any regularization-based continual learning approach such as [69, 168, 5]. Here we focus on Memory Aware Synapses [5] (MAS), our method proposed in Chapter 6, which is easy to integrate and has shown superior performance. However, we also show results with Elastic weight consolidation [69](EWC) in Section 7.4.4. Further, we ablate the components of our regularizer, both in the standard setting (Section 7.4.5) as in a setting without hard task boundaries (Section 7.4.6). Finally, we show how our regularizer improves the state-of-the-art performance on a sequence of object recognition tasks (Section 7.4.7).

7.4.1 An In-depth Comparison of Regularizers and Activation Functions for Selfless Continual Learning

We study possible regularization techniques that could lead to less interference between the different tasks in a continual learning scenario either by enforcing sparsity or decorrelation. Additionally, we examine the use of activation functions that are inspired by lateral inhibition in biological neurons that could be advantageous in continual learning. MAS [5] is used in all cases as continual learning method.

Representation Based methods:
- `L1-Rep`: to promote representational sparsity, an ℓ_1 penalty on the activations is used.
- `Decov` [28] aims at reducing overfitting by decorrelating neuron activations. To do so, it minimizes the Frobenius norm of the covariance matrix computed on the activations of the current batch after subtracting the diagonal to avoid penalizing independent neuron activations.

Activation functions:
- `Maxout` network [50] utilizes the maxout activation function. For each group of neurons, based on a fixed window size, only the maximum activation is forwarded to the next layer. The activation function guarantees a minimum sparsity rate defined by the window size.
- `LWTA` [147]: similar idea to the Maxout network except that the non-maximum activations are set to zero while maintaining their connections. In contrast to Maxout, LWTA keeps the connections of the inactive neurons which can be occupied later once they are activated without changing the previously active neuron connections.
- `ReLU` [46] the rectifier activation function (ReLU) used as a baseline here and indicated in later experiments as `No-Reg` as it represents the standard setting of continual learning on networks with ReLU. All the studied regularizers use ReLU as activation function.

Parameters based regularizers:
- `OrthReg` [125]: regularizing CNNs with locally constrained decorrelations. It aims at decorrelating the feature detectors by minimizing the cosine of the angle between the weight vectors resulting eventually in orthogonal weight vectors.
- `L2-WD`: weight decay with ℓ_2 norm [77] controls the complexity of the learned function by minimizing the magnitude of the weights.
- `L1-Param`: ℓ_1 penalty on the parameters to encourage a solution with sparse parameters.

Dropout is not considered as its role contradicts our goal. While dropout can improve

each task performance and reduce overfitting, it acts as a model averaging technique. By randomly masking neurons, dropout forces the different neurons to work independently. As such it encourages a redundant representation. As shown by [49] the best network size for classifying MNIST digits when using dropout was about 50% more than without it. Dropout steers the learning of a task towards occupying a good portion of the network capacity, if not all of it, which contradicts the continual learning needs.

Experimental setup. We use the MNIST dataset [82] as a first task in a sequence of 5 tasks, where we randomly permute all the input pixels differently for tasks 2 to 5. The goal is to classify MNIST digits from all the different permutations. The complete random permutation of the pixels in each task requires the neural network to instantiate a new neural representation for each pattern. A similar setup has been used by [69, 168, 49] with different percentage of permutations or different number of tasks. As a base network, we employ a multi layer perceptron with two hidden layers and a Softmax loss. We experiment with different number of neurons in the hidden layers $\{128, 64\}$. For `SLNID` we evaluate the effect of $\lambda_{\texttt{SLNID}}$ on the performance and the obtained sparsity in Figure 7.4a. In general, the best $\lambda_{\texttt{SLNID}}$ is the minimum value that maintains similar or better accuracy on the first task compared to the unregularized case, and we suggest to use this as a rule-of-thumb to set $\lambda_{\texttt{SLNID}}$. For λ, we have used a high λ value that ensures the least forgetting which allows us to examine the degradation in the performance on the later tasks compared to those learned previously as a result of lacking capacity. Note that better average accuracies can be obtained with tuned λ although it is not clear how to do so in continual learning settings.

All tasks are trained for 10 epochs with a learning rate 10^{-2} using SGD optimizer. ReLU is used as an activation function unless mentioned otherwise. Throughout the experiment, we used a scale σ for the Gaussian function used for the local inhibition equal to $1/6$ of the hidden layer size. For all competing regularizers, we tested different hyper parameters from 10^{-2} to 10^{-9} and report the best one.

Results: Figures 7.2 and 7.3 present the test accuracy on each task at the end of the sequence, achieved by the different regularizers and activation functions on the network with hidden layer of size 128 and 64 respectively. Clearly, in all the different tasks, the representational regularizers show a superior performance to the other studied techniques. For the regularizers applied to the parameters, `L2-WD` and `L1-Param` do not exhibit a clear trend and do not systematically show an improvement over the use of the different activation functions only. While `OrthReg` shows a consistently good performance, this performance is lower than what can be achieved by the representational regularizers. It is worth noting the `L1-Rep` yields superior performance over `L1-Param`. This observation is consistent across different sizes of the hidden layers and shows the advantage of encouraging sparsity in the activations compared to that in the parameters. Regarding the activation functions, `Maxout` and `LWTA` achieve a slightly higher performance than `ReLU`. We did not observe a significant difference between the two activation functions. However, the improvement

over ReLU is only moderate and does not justify the use of a fixed window size and special architecture design. Our proposed regularizer SLNID achieves high if not the highest performance in all the tasks and succeeds in having a stable performance. This indicates the ability of SLNID to direct the learning process towards using minimum amount of neurons and hence more flexibility for upcoming tasks.

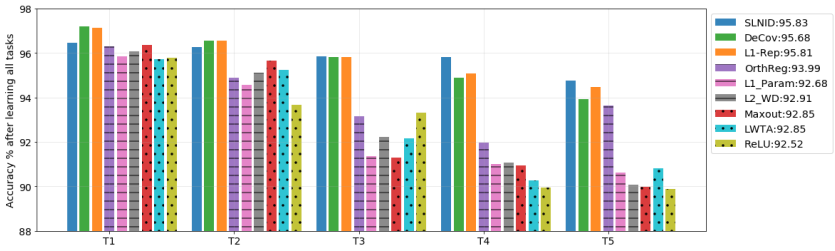


Figure 7.2: Comparison of different regularization techniques on 5 permuted MNIST sequence, hidden size=128. Representation based regularizers are solid bars, bars with lines represent parameters regularizers, dotted bars represent activation functions. Average test accuracy over all tasks is given in the legend. Representation based regularizers achieve higher performance than other compared methods including parameters based regularizers. Our regularizer, SLNID, performs the best on the last two tasks indicating that more capacity is left to learn these tasks.

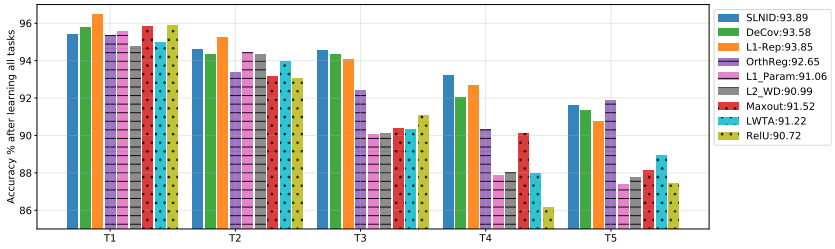
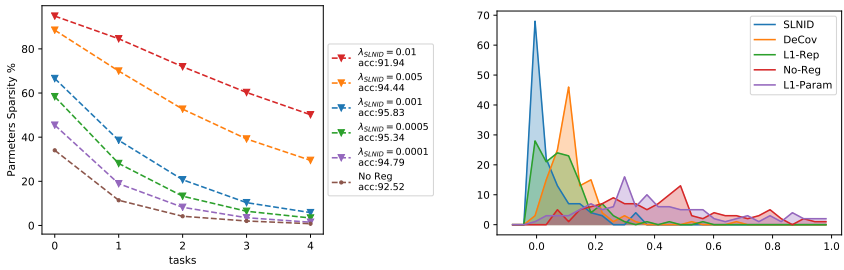


Figure 7.3: Comparison of different regularization techniques on 5 permuted MNIST sequence of tasks, hidden size=64. Representation based regularizers are solid bars, bars with lines represent parameters regularizers, dotted bars represent activation functions. See Figure 7.2 for size 128.

7.4.2 Representation sparsity & important parameter sparsity.

Here we want to examine the effect of our regularizer on the percentage of parameters that are utilized after each task and hence the capacity left for the later tasks. On the



(a) (b)

Figure 7.4: On the 5 permuted MNIST sequence, hidden layer=128, (a): percentage of unused parameters in the 1st layer using different $\lambda_{\texttt{SLNID}}$; (b): histogram of neural activations on the first task.

network with hidden layer size 128, we compute the percentage of parameters with $\Omega_{ij} < 10^{-2}$, with Ω_{ij}, the importance weights estimated and accumulated over tasks. We consider $\Omega_{ij} < 10^{-2}$ of negligible importance as in a network trained without a sparsity regularizer, $\Omega_{ij} < 10^{-2}$ covers the first 10 percentiles. Those parameters can be seen as unimportant and "free" for later tasks. Figure 7.4a shows the percentage of the unimportant (free) parameters in the first layer after each task for different $\lambda_{\texttt{SLNID}}$ values along with the achieved average test accuracy at the end of the sequence. It is clear that the larger $\lambda_{\texttt{SLNID}}$, i.e. the more neural inhibition, the smaller the percentage of important parameters. Apart from the highest $\lambda_{\texttt{SLNID}}$ where tasks couldn't reach their top performance due to too strong inhibition, improvement over the `No-Reg` is always observed.

The optimal value for $\lambda_{\texttt{SLNID}}$ seems to be the one that remains close to the optimal performance on the current task, while utilizing the minimum capacity feasible. Next, we compute the average activation per neuron, in the first layer, over all the examples and plot the corresponding histogram for `SLNID`, `DeCov`, `L1-Rep`, `L1-Param` and `No-Reg` in Figure 7.4b at their setting that yielded the results shown in Figure 7.2. `SLNID` has a peak at zero indicating representation sparsity while the other methods values are spread along the line. This seems to hint at the effectiveness of our approach `SLNID` in learning a sparse yet powerful representation and in turn maintaining a minimal interference between tasks.

7.4.3 10 Task Sequences on Cifar-100 and Tiny ImageNet

While the previous section focused on learning a sequence of tasks with completely different input patterns and same objective, we now study the case of learning different categories of one dataset.

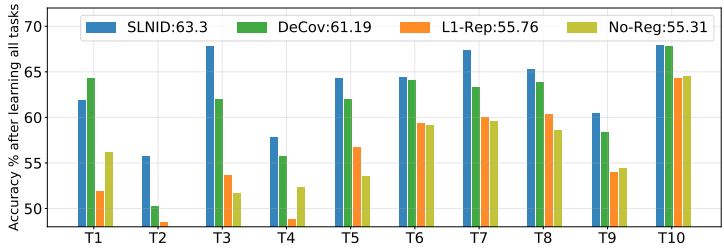


Figure 7.5: Comparison of different regularization techniques on a sequence of ten tasks from Cifar split. The legend shows average test accuracy over all tasks. Simple L1-norm regularizer (L1-Rep) doesn't help in such more complex tasks. Our regularizer `SLNID` achieves an improvement of 2% over Decov and 8% compared to No-Reg.

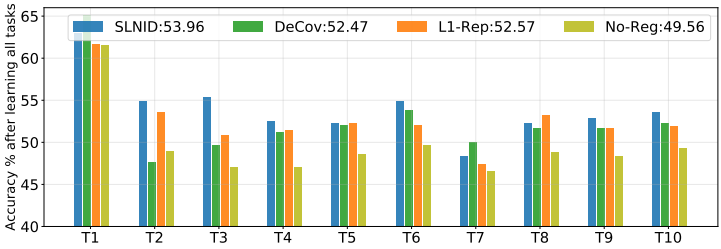


Figure 7.6: Comparison of different regularization techniques on a sequence of ten tasks from Tiny ImageNet split. The legend shows average test accuracy over all tasks. Our regularizer `SLNID` achieves the best performance.

For this we split the Cifar-100 and the Tiny ImageNet [163] dataset into ten tasks, respectively. We have 10 and 20 categories per task for Cifar-100 and Tiny ImagNet, respectively. For Cifar-100, as a base network, we use a network similar to the one used by [168] but without dropout. We evaluate two variants with hidden size $N = \{256, 128\}$.

Throughout the experiment, we again used a scale σ for the Gaussian function equal to $1/6$ of the hidden layer size. We train the different tasks for 50 epochs with a learning rate of 10^{-2} using SGD optimizer. For Tiny ImageNet dataset [163], as a base network, we use a variant of VGG [143]. For architecture details, please refer to Table 7.1.

We compare the top competing methods from the previous experiments, `L1-Rep`, `DeCov` and

Layer	# filters/neurons
Convolution	64
Max Pooling	-
Convolution	128
Max Pooling	-
Convolution	256
Max Pooling	-
Convolution	256
Max Pooling	-
Convolution	512
Convolution	512
Fully connected	500
Fully connected	500
Fully connected	20

Table 7.1: The network architecture used in Tiny ImageNet experiment.

our SLNID, and No−Reg as a baseline, ReLU in previous experiment. Similarly, MAS [5] is used in all cases as continual learning method. Figures 7.5 and 7.6 show the performance on each of the ten tasks at the end of the sequence. For both datasets, we observe that our SLNID performs overall best. L1−Rep and DeCov continue to improve over the non regularized case No−Reg. These results confirm our proposal on the importance of sparsity and decorrelation in continual learning.

7.4.4 SLNID with EWC [69]

We have shown that our proposed regularizer SLNID exhibits stable and superior performance on the different tested networks when using MAS as continual learning method. To prove the effectiveness of our regularizer regardless of the used regularization based method, we have tested SLNID on the 5 tasks permuted MNIST sequence in combination with Elastic Weight Consolidation (EWC [69]). Figures 7.7 and 7.8 compare the test accuracy of each task achieved by our SLNID with EWC and No−Reg (here indicating EWC without regularization) on a network of size 128 and 64 respectively. We obtained a boost in the average performance at the end of the learned sequence equal to 2.8% on the network with hidden layer size 128 and a boost of 3.6% with hidden layer size 64. It is worth noting that with both MAS and EWC, SLNID was able to obtain better accuracy using a network with a 64-dimensional hidden size than when training without regularization No−Reg on a network of double that size (128), indicating that SLNID allows to use neurons much more efficiently.

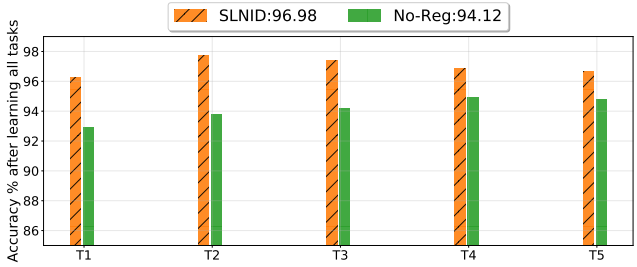


Figure 7.7: Comparison of SLNID, with EWC [69], and No−Reg, EWC alone with no sparsity regularizer. Figure shows at the end of permuted MNIST sequence, each task accuracy on a network with 128 neurons in the hidden layer.

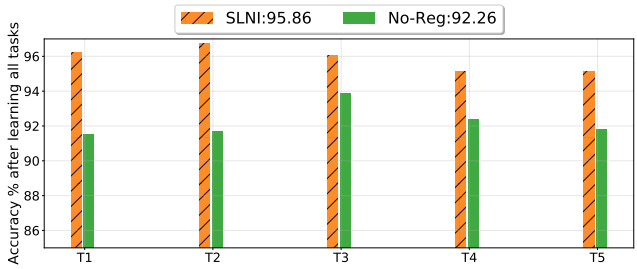


Figure 7.8: Comparison of SLNID, with EWC [69], and No-Reg, EWC alone with no sparsity regularizer. Figure shows at the end of permuted MNIST sequence, each task accuracy on a network with 64 neurons in the hidden layer.

7.4.5 Ablation Study

Our method can be seen as composed of three components: the neural inhibition, the locality relaxation and the neuron importance integration. To study how these components perform individually, we consider the following variants:

- SNI : our regularizer without spatial locality and neuron importance.
- SNID: our regularizer without spatial locality.
- SLNI: our regularizer without neuron importance.
- SLNID: our full regularizer.

Table 7.2 reports the average accuracy at the end of the Cifar-100 and permuted MNIST sequences for each variant. As we explained in Section 7.3, when tasks have completely different input patterns, the neurons that were activated on the previous task examples will not fire for new task samples and exclusion of important neurons is not mandatory. However, when sharing is present between the different tasks, a term to prevent our regularizer from causing any interference is required. This is manifested in the reported results: for permuted MNIST, all the variants work nicely alone, as a result of the simplicity and the disjoint nature of this sequence. However, in the Cifar-100 sequence, the integration of the neuron importance in the SNID and SLNID regularizers exclude important neurons from the inhibition, resulting in a clearly better performance. The locality in SLNID improves the performance in the Cifar sequence, which suggests that a richer representation is needed and multiple active neurons should be tolerated.

	Permuted MNIST		Cifar	
h-layer dim.	128	64	256	128
Joint Training*	97.30	96.80	70.99	71.95
No-Reg	92.67	90.72	55.06	55.3
SNI	95.79	**94.89**	55.30	55.75
SNID	95.90	93.82	61.00	60.90
SLNI	**95.95**	94.87	56.06	55.79
SLNID	95.83	93.89	**63.30**	**61.16**

Table 7.2: SLNID ablation. Average test accuracy per task after training the last task in %. * denotes that Multi-Task Joint Training violates the continual learning scenario as it has access to all tasks at once and thus can be seen as an upper bound.

7.4.6 Continual Learning without Hard Task Boundaries

In the previous experiments, we considered the standard task incremental setting as in [85, 168, 5, 136], where at each time step we receive a task along with its training data and a new classification layer is initiated for the new task, if needed. Here, we are interested in a more realistic scenario where the data distribution shifts gradually without hard task boundaries. To test this setting, we use the Cifar-100 dataset. Instead of considering a set of 10 disjoint tasks each composed of 10 classes, as in the previous experiment (Section 7.4.3), we now start by sampling with high probability $(2/3)$ from the first 10 classes and with low probability $(1/3)$ from the rest of the classes. We train the network (same architecture as in Section 7.4.3) for a few epochs and then change the sampling probabilities to be high $(2/3)$ for classes $11-20$ and low $(1/3)$ for the remaining classes. This process is repeated until sampling with high probability from the last 10 classes and low from the rest. We use one shared classification layer throughout and estimate the importance weights and the neurons importance after each training step (before changing the sampling probabilities). We consider 6 variants: our SLNID, the ablations SLNI and without regularizer No-Reg, as in Section 7.4.5, as well each of these three trained without the MAS importance weight regularizer, denoted as w/o MAS. Table 7.3 presents the accuracy averaged over the ten groups of ten classes, using each group model (i.e. the model trained when this group was sampled with high probability) in the top block and the average accuracy on each of the ten groups at the end of the training (middle and bottom block). We can deduce the following: 1) SLNID improves the performance considerably

Method	Avg.acc -tasks models
No-Reg w/o MAS	69.20%
SLNI w/o MAS	72.14%
SLNID w/o MAS	73.03%
No-Reg	66.88%
SLNI	71.32%
SLNID	72.33%
Method	**Avg.acc-last model**
No-Reg w/o MAS	65.15%
SLNI w/o MAS	63.54%
SLNID w/o MAS	70.75%
No-Reg	66.33%
SLNI	64.50%
SLNID	70.94%

Table 7.3: No tasks boundaries test case on Cifar-100. Top block, avg. acc on each group of classes using each group model. Bottom block, avg. acc. on each group at the end of the training.

(by more than 4%) even without importance weight regularizer. 2) In this scenario without hard task boundaries there is less forgetting than in the scenario with hard task boundaries studied in Section 7.4.3 for Cifar (difference between rows in top block to corresponding rows in middle block). As a result, the improvement obtained by deploying the importance weight regularizer is moderate: at 70.75%, SLNID w/o MAS is already better than No-Reg reaching 66.33%. 3) While SLNI w/o MAS improves the individual models performance (72.14% compared to 69.20%), it fails to improve the overall performance at the end of the sequence (63.54% compared to 65.15%), as important neurons are not excluded from the penalty and hence they are changed or inhibited leading to tasks interference and performance deterioration.

7.4.7 Comparison with the State of the Art

To compare our proposed approach with the different state-of-the-art continual learning methods, we use a sequence of 8 different object recognition tasks, introduced in Chapter 6. The sequence starts from AlexNet [76] pretrained on ImageNet [132] as a base network, following the setting of Chapter 6. We compare against the following: *Learning without Forgetting* [85] (LwF), *Encoder Based Lifelong Learning* (EBLL) (see Chapter 5), *Incremental Moment Matching* [83] (IMM), *Synaptic Intelligence* [168] (SI) and sequential finetuning (FineTune), in addition to the case of MAS [5] alone, i.e. our No-Reg before. Compared methods were run with the exact same setup as in Chapter 6. For our regularizer, we disable dropout, since dropout encourages redundant activations which contradicts our regularizer's role. Also, since the network is pretrained, the locality introduced in SLNID may conflict with the already pretrained activations. For this reason, we also test SLNID with randomly initialized fully connected layers. Our regularizer is applied with MAS as a continual learning method. Table 7.4 reports the average test accuracy at the end of the sequence achieved by each method. SLNID improves even when starting from a pretrained network and disabling dropout. Surprisingly, even with randomly initialized fully connected layers, SLNID improves 1.8% over the state of the art using a fully pretrained network.

Method	Average accuracy
Finetune	32.67%
LWF [85]	49.49%
EBLL	50.29%
IMM [83]	43.4%
SI [168]	50.49%
EWC [69]	50.00%
MAS	52.69%
SLNID-fc pretrained	53.77%
SLNID-fc randomly initialized	**54.50%**

Table 7.4: 8 tasks object recognition sequence. Average test accuracy per task after training the last task.

7.4.8 Spatial Locality Test

To avoid penalizing all the active neurons, our SLNID weights the correlation penalty between each two neurons based on their spatial distance using a Gaussian function. We want to visualize the effect of this spatial locality on the neurons activity. To achieve this, we have used the first 3 tasks of the permuted MNIST sequence as a test case and visualized the neurons importance after each task. This is done using the network of hidden layer size 64. Figure 7.9, Figure 7.10 and Figure 7.11 show the neurons importance after each task. Figure 7.12, Figure 7.13 and Figure 7.14 show the neurons importance after each task sorted in descending order according to the first task neuron importance. The left column is without locality, i.e. SNID, and the right column is SLNID. Blue represents the first task, orange the second task and green the third task. When using SLNID, inhibition is applied in a local manner allowing more active neurons which could potentially improve the representation power. When learning the second task, new neurons become important regardless of their closeness to first task important neurons as those neurons are excluded from the inhibition. As such, new neurons are becoming active as new tasks are learned. For SNID on the other hand, all neural correlation is penalized in the first task and for later tasks. Very few neurons are able to become active and important due to the strong global inhibition, where previous neurons that are excluded from the inhibition are easier to be re-used.

7.5 Summary

In this chapter, we have studied the problem of continual learning using a network with fixed capacity – a prerequisite for a scalable and computationally efficient solution. A key insight of our approach is that in the context of continual learning (as opposed to other contexts where sparsity is imposed, such as network compression or avoiding overfitting), sparsity should be imposed at the level of the representation rather than at the level of the network parameters. Inspired by lateral inhibition in the mammalian brain, we impose sparsity by means of a new regularizer that decorrelates nearby active neurons. We integrate this in a model which *learns selflessly* a new task by leaving capacity for future tasks and at the same time *avoids forgetting* previous tasks by taking into account neurons importance.

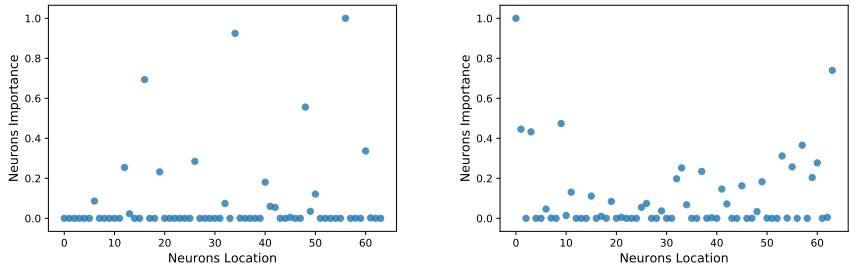


Figure 7.9: First layer neuron importance after learning the first task (blue). Left: SNID, Right: SLNID. More active neurons are tolerated in SLNID.

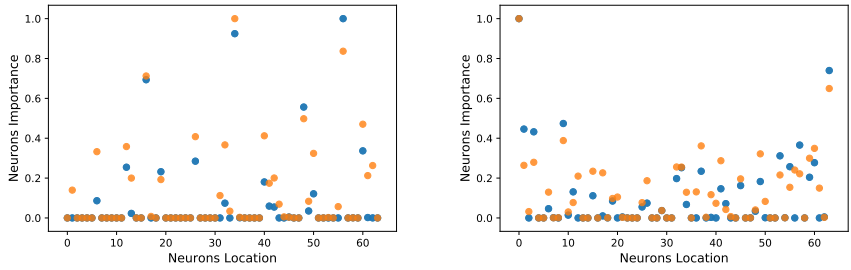


Figure 7.10: First layer neuron importance after learning the second task (orange), superimposed on Figure 7.9. Left: SNID, Right: SLNID. SLNID allows new neurons, especially those that were close neighbours to previous important neurons, to become active and to be used for the new task. SNID penalizes all unimportant neurons equally. As a result, previous neurons are adapted for the new tasks and less new neurons are getting activated.

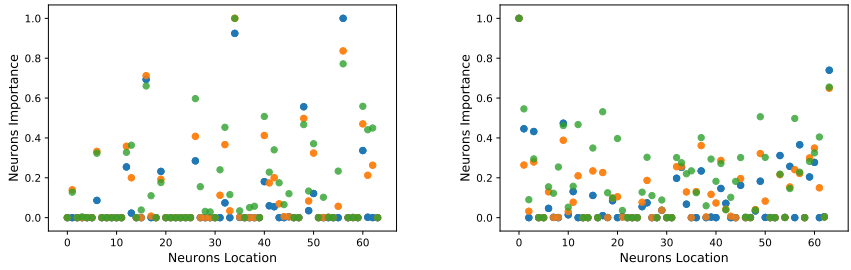


Figure 7.11: First layer neuron importance after learning the third task (green), superimposed on Figure 7.10. Left: SNID, Right: SLNID. SLNID allows previous neurons to be re-used for the third task. It avoids changing the previous important neurons by adding new neurons. For SNID, very few neurons are newly deployed. The new task is learned mostly by adapting previous important neurons.

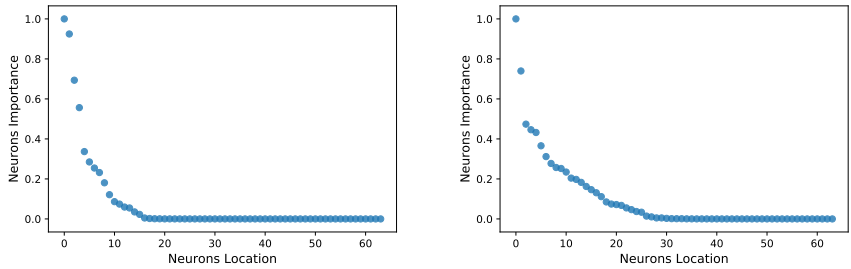


Figure 7.12: First layer neuron importance after learning the first task, sorted in descending order according to the first task neuron importance (blue). Left: SNID, Right: SLNID. More active neurons are tolerated in SLNID.

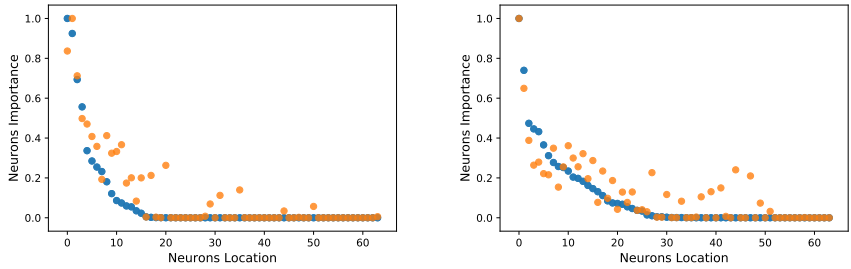


Figure 7.13: First layer neuron importance after learning the second task sorted in descending order according to the first task neuron importance (orange), superimposed on top of Figure 7.12. Left: SNID, Right: SLNID. SLNID allows new neurons to become active and be used for the new task. SNID penalizes all unimportant neurons equally and hence more neurons are re-used then initiated for the first time.

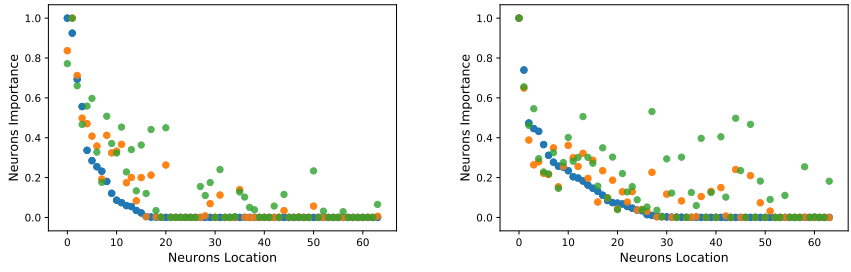


Figure 7.14: First layer neuron importance after learning the third task sorted in descending order according to the first task neuron importance (green), superimposed on top of Figure 7.13. Left: SNID, Right: SLNID. SLNID allows previous neurons to be re-used for the third task while activating new neurons to cope with the needs of the new task. For SNID, very few neurons are newly deployed while most previous important neurons for previous tasks are re-adapted to learn the new task.

Chapter 8

Regularization based Online Continual Learning

Methods proposed in the literature towards continual learning typically operate in a task incremental setting [83, 85, 168, 69, 96]. A sequence of tasks is learned, one at a time, with all the data of the current task available but not of previous or future tasks. Task boundaries and identities are known at all times. This setup, however, is less likely in various practical applications. Therefore we investigate how to transform task incremental setting to a more general online continual scenario. We develop a system that keeps on learning over time in a streaming fashion, with data distributions gradually changing and without the notion of separate tasks. To this end, we build on our importance weight regularizer, MAS (Chapter 6), and show how it can be made online by providing a protocol to decide i) when to update the importance weights, ii) which data to use to update them, and iii) how to accumulate the importance weights at each update step. Experimental results show the validity of the approach in the context of two applications: (self-)supervised learning of a face recognition model by watching soap series and learning a robot to avoid collisions.

This is a joint work with Klaas Kelchtermans. We have developed closely the approach. While I focused on the soap series experiments, Klaas focused on the experiments of the robot navigation. This work was published as an article in CVPR 2019 [8].

8.1 Introduction

In machine learning, one of the most basic paradigms is to clearly distinguish between a training and testing phase. Once a model is trained and validated, it switches to a test mode: the model gets frozen and deployed for inference on previously unseen data, without ever making changes to the model parameters again. This setup assumes a "static" world, with no distribution shifts over time. Further, it assumes a static task specification, so no new requirements in terms of output (e.g. novel category labels) or new tasks added over time. Such strong division between training and testing makes it easier to develop novel machine learning algorithms, yet is also very restrictive.

Inspired by biological systems, continual learning aims at breaking this strong barrier between the training and testing phase. As we have shown in previous chapters, many continual learning methods have been developed with the goal of training a shared model on one task at a time with significant progress achieved. However, this specific setup highly depends on knowing the task boundaries. These boundaries indicate good moments to consolidate knowledge, namely after learning a task. Moreover, data can be shuffled within a task so as to guarantee i.i.d. assumption. In an online setting, on the other hand, data needs to be processed in a streaming fashion and data distributions might change gradually.

In this chapter, we aim at overcoming this requirement of hard task boundaries. In particular, we investigate how methods proposed for task incremental setting can be generalized to an online setting. This requires a protocol to determine when to consolidate knowledge. Moreover, we investigate the effect of keeping a small buffer with difficult samples. For the latter, we take inspiration from the field of reinforcement learning, namely experience replay [106], although using much smaller replay buffers, unlike very recent work of Rolnick et al. [127].

Task incremental setting has mostly been studied for image classification [85, 6, 83, 168, 120]. Whenever the learner arrives at a new task, that is when learning on the previous task has converged, a standard procedure is to extend the output layer of the network with additional "heads" for the new task's categories. Instead, the output of the network considered here is fixed. In the first application, learning to recognize faces, we cope with a varying number of categories by using an embedding rather than class predictions. In the second application, learning a lightweight robot to navigate without collisions, it is not the output labels that change over time but rather the environment. For both applications, data are processed in a streaming fashion. This is challenging, since the data are not i.i.d. distributed, causing samples within one batch to be unbalanced.

The contributions of this chapter are as follows: i) We were the first to extend the task incremental setting to free and unknown task boundaries in an online continual learning

scenario; ii) We develop protocols to integrate our importance weight regularizer, MAS, in this online continual learning setting; iii) Our experiments on face recognition from T.V. series and on monocular collision avoidance prove the effectiveness of our method in handling the distribution changes in the streaming data and stabilizing the online learning behavior, resulting in knowledge accumulation rather than catastrophic interference and improved performance in all the test cases.

In the following we discuss related work (Section 8.2). We then describe our online continual learning approach in Section 8.3. We validate our system in Section 8.4 and end with discussion and conclusion in Section 8.5.

8.2 Related Work

Online Learning: As we have have explained in the introduction (Chapter 1, Section 1.2) online learning addresses the learning problem over a stream of data instances sequentially.

A first set of online learning algorithms consists of different techniques designed to learn a linear model [53, 33, 148, 61]. Online learning with kernels [70] extends this line of work to non-linear models, but the models remain shallow and their performance lags behind the modern deep neural networks. Some recent works include [135, 119], start from a small network and then adapt the capacity by adding more neurons as new samples arrive. Typically, online learning methods don't consider cases where the data generating distribution is changing overtime as in continual learning. As such, they are vulnerable to catastrophic forgetting once a distribution change occurs.

In terms of applications, the work of Pernici *et al.* [115, 114] is similar to our first application scenario. They learn face identities in a self-supervised fashion via temporal consistency. They start from the VGG face detector and descriptor, and use a memory of detected faces. In contrast, we start from a much weaker pretrained model (not face-specific), and update the model parameters over time while they do not.

Continual Learning: While all continual learning methods to the date of development (2018/2019) [168, 69, 5, 83, 164, 85, 120, 122, 145] follow the task incremental learning setup, a special mention here goes to Gradient Episodic Memory (GEM) [92], as it moves a step forward towards the online setting. GEM assumes that the learner receives examples one by one but simplifies the scenario to locally i.i.d. drawn samples from a distribution of a given task. It further assumes that a task identifier is given and used it to build an episodic memory of most recent samples for each seen task. This memory is then exploited to prevent forgetting when learning new tasks. In our work, we use a very small buffer, around 100 samples, to guarantee better learning of hard

samples. In the same time, this small buffer is deployed to estimate the parameters importance once a stable phase of learning has been reached.

The concept of replay buffer is often used in Deep Reinforcement Learning (DRL). However a crucial difference is that in both old and recent DRL works the replay buffer typically contains up to 1M samples corresponding to over 100 days of experience [106, 56]. A common DRL technique, known as "prioritized sweeping", is to sample experiences with large errors more often than others [107]. Our approach follows a similar strategy, however, to select which samples are going to be stored in a small buffer.

8.3 Method

Our goal is to design a training method for task-free online continual learning. In this setting where data is streaming and the distribution is shifting gradually, it is unclear whether current continual learning methods can be applied and how.

After studying some of the existing methods, we find out that our importance weight regularizer (MAS), see Chapter 6, is indeed one of the most promising methods in this respect. It enjoys the following favorable characteristics.

1. *Constant memory*: it only stores an importance weight for each parameter in the network avoiding an increase in memory consumption over time.
2. *Problem agnostic*: it can be applied to any task and is not limited to classification. In particular, we can use it with an embedding as output, avoiding the need to add extra "heads" for new outputs over time.
3. *Fast*: it only needs one backward pass to update the importance weights. During training, the gradients of the imposed penalty are simply the change that occurs on each parameter weighted by its importance. Therefore, the penalty gradients can be added locally and do not need a backpropagation step.
4. *Top performance*: MAS shows superior performance to other importance weight regularizers [60].

In order to deploy MAS in an online continual learning scenario, we need to determine i) when to update the importance weights, ii) which data to use to update the importance weights, and iii) how to accumulate the importance weights at each update step. We first introduce the considered online continual learning setup, then explain our training procedure under this setup.

Setup: Online continual learning represents the general and desired setting of continual learning as we have explained in the introduction (Chapter 1, Section 1.1). Here is a gentle reminder: We assume an infinite stream of data and a supervisory or self-supervisory signal that is generated based on few consecutive samples. At each time step t, the system receives a few consecutive samples along with their generated labels $\{x_t, y_t\}$ drawn non i.i.d. from a current distribution Q. Moreover, the distribution Q could itself experience sudden or gradual changes. The system is unaware of when these distribution changes are happening. The goal is to continually learn and update a function f that minimizes the prediction errors on previously seen and future samples. In other words, it aims at continuously updating and accumulating knowledge.

We formulate the learning objective of an online system as follows. Given an input model with parameters θ, the system at each time step reduces the empirical risk based on the recently received samples and a small buffer $\mathcal{B} = (X_\mathcal{B}, Y_\mathcal{B})$ composed of updated hard samples. The objective function is then:

$$J(\theta) = \ell(f(x_t; \theta), y_t) + \ell(f(X_\mathcal{B}; \theta), Y_\mathcal{B}) \tag{8.1}$$

Due to the strong non-i.i.d. conditions and the very low number of samples used for the gradient step, the system is vulnerable to catastrophic interference between recent samples and previous samples and faces difficulty in accumulating the knowledge over time.

As we explained in Chapter 6 in a traditional task incremental setting, MAS estimates an importance weight for each network parameter after each training phase (task). The importance weights are estimated using training or test data from the task at hand. When learning a new task, changes to important parameters are penalized.

When to update importance weights: In the case of a task incremental setting where tasks have predefined boundaries, importance weights are updated after each task, when learning has converged. In the online case, the data is streaming without knowledge of a task's start or end (i.e. when distribution shifts occur). We need a mechanism to determine when to update the importance weights. For this, we look at the surface of the loss function.

By observing the loss, we can derive some information about the data presented to the system. When the loss decreases, this indicates that the model has learned some meaningful new knowledge from those seen samples. Yet the loss does not systematically decrease all the time. When new samples are received that are harder or contain different concepts or input patterns than what was presented to the learner before, the loss may increase again. In these cases, the model has to update its knowledge, while minimally interfering with what has been learned previously.

We can conclude that plateaus (i.e. areas with consistent small values) in the loss function indicate stable learning regimes, where the model is confidently predicting the

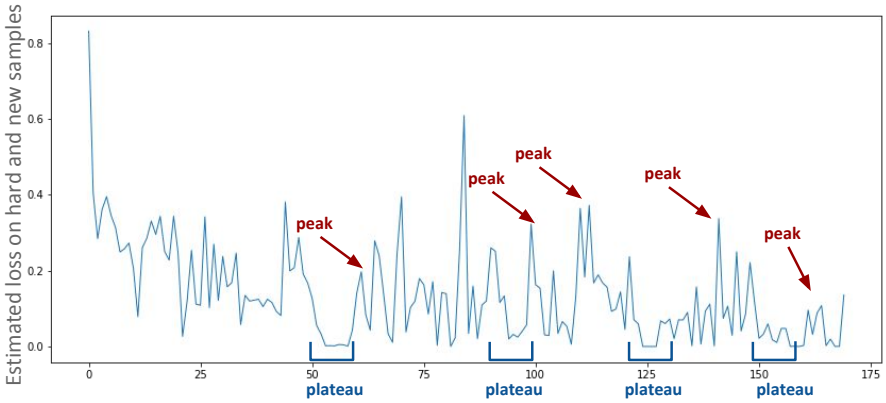


Figure 8.1: By detecting "plateaus" and "peaks" in the loss surface our method decides when to update the importance weights. Figure corresponds to the BBT experiment, see Section 8.4.2; x-axis represents update steps

current labels, see Figure 8.1. Whenever the model is in such a stable area, it is a good time to identify the parameters that are important for the currently acquired knowledge. When learning new, "different" samples the model will then be encouraged to preserve this knowledge. This should allow the model to accumulate knowledge over time rather than replacing previously learned bits of knowledge.

Detecting plateaus in the loss surface: To detect these plateaus in the loss surface, we use a sliding window over consecutive losses during training. We monitor the mean and the variance of the losses in this window and trigger an importance weight update whenever they are both lower than a given threshold. We do not keep re-estimating importance weights: we only re-check for plateaus in the loss surface after observing a peak. Peaks are detected when the loss window mean becomes higher than 85% of a normal distribution estimated on the loss window of the previous plateau - that is when $\mu(L_{win}) > \mu_L^{old} + \sigma_L^{old}$ where μ_L^{old} and σ_L^{old} are the statistics of the previously detected plateau. This accounts for the continuous fluctuations in the loss function in the online learning and detects when significantly harder samples are observed.

A small buffer with hard samples: In a task incremental learning setup, importance weights are estimated on the training/test data of a task after training. This is not an option for online learning, as storing all the previous data violates the condition of our setup. On the other hand, using only the most recent sequence of samples would lead to misleading estimates as these few consecutive samples might not be representative and hence do not capture the acquired knowledge correctly. We propose to use a small buffer of hard samples. The buffer of hard samples is updated at each learning step

by keeping the samples with highest loss among the new samples and the current buffer. This is important as previous samples cannot be revisited. Hence, it also gives the system the possibility to re-process those hard samples and adjust its parameters towards better predictions. In addition, it provides a better estimate of the gradient step by averaging over the recent and hard samples. More importantly, the hard buffer represents better the previous history than the few recent samples, hence allows for a better identification of importance weights. Note that we don't consider the presence of outliers. In the context of online continual learning, the only assumption we make is that the label we receive for each training sample is valid. Handling outliers is left for further investigation.

Accumulating importance weights: As we frequently update the importance weights, simply adding the new estimated importance values to the previous ones would lead to very high values and exploding gradients. Instead, we maintain a cumulative moving average of the estimated importance weights. Note, one could deploy a decaying factor that allows replacing old knowledge in the long term. However, in our experiments a cumulative moving average showed more stable results.

After updating the importance weights, the model continues the learning process while penalizing changes to parameters that have been identified as important so far. As such our final objective function is:

$$J(\theta) = \ell(f(x_t;\theta), y_t) + \ell(f(X_{\mathcal{B}};\theta), Y_{\mathcal{B}}) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta^*_{ij})^2 \tag{8.2}$$

where θ^* are the parameters values at the last importance weight update step. Algorithm 3 summarizes the different steps of the proposed continual learning system.

8.4 Experiments

As a proof-of-concept, we validate our proposed method on a simple synthetic experiment. Later, we evaluate the method on two applications with either weak or self-supervision. First, we learn actor identities from watching soap series. The second application is robot navigation. In both cases, data are streaming and online continual learning is a key factor.

8.4.1 Synthetic Experiment

We want to validate our method on a controlled environment where the changes in the training data are well identified. For this reason, we constructed a binary classification problem with points in 4D in/out of the unit sphere. We consider a

Algorithm 3 Online Continual Learning

Input:$\delta_\mu, \delta_\sigma, \mathcal{S}, \lambda$ ▷ Loss window statistics, number of gradient steps.
Initialize: $\mathcal{B} = \{\}, \mathcal{W} = \{\}, \Omega = \vec{0}, \theta^* = \vec{0}, \mu_L^{old} = 0, \sigma_L^{old} = 0, \mathcal{P} = 1$
repeat
  Receive M recent samples X, Y
  for s in $\mathcal{S}$ **do**
    $J(\theta) = \ell_\theta(X, Y) + \ell_\theta(X_\mathcal{B}, Y_\mathcal{B}) + \lambda \sum_{i,j} \Omega_{ij}(\theta_{ij} - \theta^*_{ij})^2$
    if $s = 1$ **then**
      $\mathcal{W} \leftarrow$ update$(\mathcal{W}, \ell_\theta(X, Y), \ell_\theta(X_\mathcal{B}, Y_\mathcal{B}))$ ▷ Update the loss window.
    end if
    $\theta \leftarrow$SGD$(J(\theta))$
  end for
  if $\mathcal{P}$ & $\mu(\mathcal{W}) < \delta_\mu$ & $\sigma(\mathcal{W}) < \delta_\sigma$ **then**
    $\Omega \leftarrow$ update$(\Omega, \theta, (X_\mathcal{B}, Y_\mathcal{B}))$
    $\theta^* \leftarrow \theta$
    $\mu_L^{old} = \mu(\mathcal{W}), \sigma_L^{old} = \sigma(\mathcal{W})$
    $\mathcal{W} = \{\}, \mathcal{P} = 0$
  end if
  if $\mu(\mathcal{W}) > \mu_L^{old} + \sigma_L^{old}$ **then**
    $\mathcal{P} = 1$
  end if
  $(X_\mathcal{B}, Y_\mathcal{B}) \leftarrow$ update$((X_\mathcal{B}, Y_\mathcal{B}), (X, Y), \ell_\theta(X, Y))$

simple two tasks scenario where each task corresponds to a quadrant in the unit sphere. A simple hypothesis of a hyper plane separating in/out points of the first quadrant will be orthogonal to an optimal plane solving the second quadrant task. As a result, catastrophic forgetting is expected to happen if no mechanism is deployed to prevent changing the learned hypothesis drastically.

To validate the effect of our approach, coined `Online Continual`, we consider two baselines:

- `Online` a model trained online using a hard buffer with the objective of Equation 8.1
- `Online No Hard` the online trained model but without the aid of the hard buffer.

All methods are trained online with data from the first quadrant and then task switches to the second quadrant still online without providing the compared methods with this shift information. Figure 8.2 depicts the methods predictions near the decision boundary in the two quadrants at the end of training the second quadrant.

As expected the forgetting is sever and apparent in the baselines predictions. The hard buffer results in better learning (higher total test accuracy) but doesn't stand against forgetting alone. Our method `Online Continual` with MAS deployed succeeds

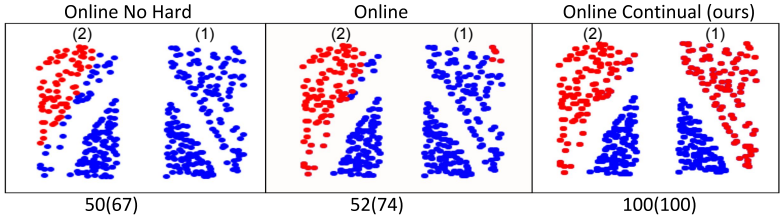


Figure 8.2: Synthetic experiment: Predictions on first (right) and second (left) quadrants after training on the second quadrant. Test accuracy (%) on the first quadrant (total test accuracy on both quadrants(%)) shown beneath the figure.

in perfectly predicting the in/out points of the unit sphere from both the first and the second quadrants. Our method has updated the importance weights after learning a good hypothesis on first quadrant and used the importance weights to prevent forgetting when data shifted to the second quadrant.

Having proven the effectiveness of our method on a simple synthetic experiment, we move to test more complex situations with streaming data and gradually shifting distributions.

8.4.2 Continual Learning by Watching Soap Series

Here, we assume that an intelligent agent is watching soap series episodes and learns to differentiate between the faces of the different actors. The agent is equipped with a face detector module that is detecting faces online and a multi-object face tracker. In the case of weak supervision, we assume there is an annotator telling the agent whether two consecutive tracks are of the same identity or not. For the self-supervised case, we use the fact that if two faces are detected in the same image then their tracks must belong to two different actors. In both cases, faces within a track considered to belong to the same actor.

Setup: We start from AlexNet [74] architecture with the convolutional layers pre-trained on ImageNet [76] and the fully connected layers initialized randomly. The output layer is of size 100. Since the input consists of two tracks of two different identities, we use the triplet margin loss [14] which has been shown to work well in face recognition applications. This has the additional advantage that we don't need to know all the identities beforehand and new actors can be added as more episodes are watched.

Dataset: We use the actor labelling dataset from [7], specifically 6 episodes of The Big Bang Theory (BBT), 4 episodes of Breaking Bad (BB), and one episode of Mad

Figure 8.3: Four example images for each soap series, from left to right: Big Bang Theory (BBT), Breaking Bad (BB) and Mad Men (MM).

Men (MM)[1]. Note that for BB and MM, the episodes were further split into a total of 22 and 5 chunks, respectively. For each episode we use the frames, detected faces and tracks along with track labels from [7]. Tracks are processed in chronological order, imitating the setting where tracks are extracted in an online fashion while watching the soap series. As a result, the data are clearly non-i.i.d.. Figure 8.3 shows 4 example frames for each of the different soap series: Big Bang Theory, Breaking Bad and Mad Men. These examples demonstrate the scene diversity and large variance in imaging conditions. Breaking Bad is more actor-centric with a majority of the frames showing only the main character.

For the supervised setup, every tenth/fifth track is held out as test data in BBT/BB respectively since the latter has more tracks, 339 tracks BBT compared to 3941 BB. All the other tracks are used for training. As we only have one episode for MM, we decided not to use it for the supervised setup.

For the self-supervised setup, BB turned out to be unsuitable, given that it is an actor centric series with a large majority of the scenes focusing on one actor. To still have results on two series, we do report also on MM in this case, in spite of it being only one episode. Further, the original tracks provided by [7] were quite short (an average of 8/22 faces per track in BBT/MM). Since this is problematic for the self-supervised setting, we use a simple heuristic based on the distance between the faces embedding (based on AlexNet pretrained on ImageNet) to merge adjacent tracks belonging to the same actor. Table 8.1 states the total number of training tracks and images in each dataset regarding both weak supervision and self supervision scenarios.

Training: Whenever two tracks are encountered belonging to different actors, a training step is performed using the detected faces (one face every 5 frames). If the two tracks contain more than 100 faces, a random sampling step is performed. We use a hard buffer size of 100 triplets and a fixed loss window size of 5. A few gradient descent steps

[1] Unfortunately, there was an issue with the labels for the other episodes of Mad Men, which prevented us from using these.

are performed at each training step (2-3 for the supervised setting, 10-15 for the self-supervised one). We use SGD optimizer with a learning rate of 10^{-4}. Hyperparameters were set based on the first BBT episode, please refer to Table 8.2 for more details.

Test: To test the accuracy of the trained model on recognizing the actors in the soap series, we use 5 templates of each actor selected from different episodes. We then compute the Euclidean distance of each test face to the templates, based on the learned representation, and assign the input face to the identity of the template that is closest.

T.V. series	self-supervision	# of tracks	# of training images
BBT	No	339	2,633
BBT	Yes	223	588
BB	No	1,387	5,190
MM	Yes	144	2,763

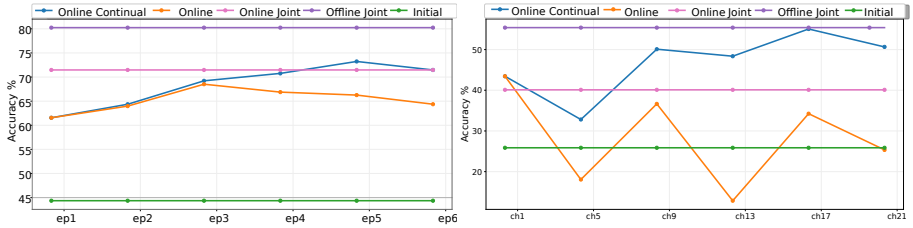
Table 8.1: Statistics of the deployed T.V. series datasets in both supervision cases.

	Soap Series (Sec. 8.4.2)	Corridor (Sec. 8.4.3)	Turtlebot (Sec. 8.4.4)
Architecture	AlexNet	Tiny v2	Tiny v2
Initialization	ImageNet	random	random
Learning rate	0.0001	0.01	0.01
Optimizer	SGD	SGD	SGD
Hard Buffer Size $\|\mathcal{B}\|$	100	40	30
Regularization Weight (λ)	100	0.5	0.5
Mean Loss Thresholdδ_μ	0.3	0.5	0.5
Variance Loss Threshold δ_σ	0.1	0.1	0.02
Loss Window Length $\|\mathcal{W}\|$	5	5	5

Table 8.2: Hyperparameters used in the different experiments of this chapter.

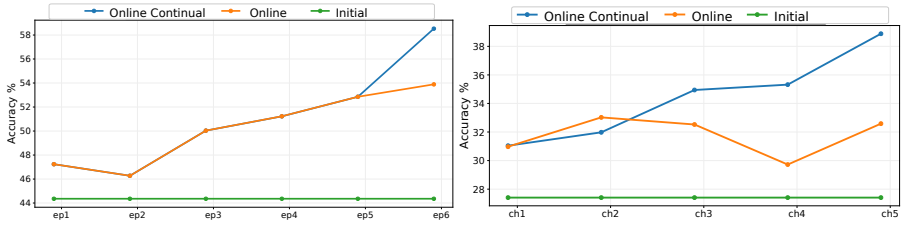
Baselines: To estimate the benefit of our system, `Online Continual`, we compare it against the following baselines:

1. `Initial` : the pretrained model, i.e. before training on any of the episodes.
2. `Online` : a model trained in the explained online setting but without our importance weight regularizer, MAS.
3. `Online Joint` : a model trained online, again without MAS regularization, but with shuffled tracks across episodes to obtain i.i.d. samples.
4. `Offline Joint` : a model that differs from Online Joint by going multiple epochs over the whole data. This stands as an upper bound.



(a) (b)

Figure 8.4: Accuracy on test data at the end of each episode for BBT (a) and after chunks 1,5,9,13,17 and 21 of BB (b), under weak supervision.



(a) (b)

Figure 8.5: Self-supervised setting: accuracy on all faces of BBT after each episode (a) and of MM after each of the 5 chunks (b).

Weak Supervision Results

Figure 8.4 (a) shows the actor recognition accuracy evaluated on all the test data of BBT, at the end of each episode. Initially, `Online` (orange) obtains an increase of 20% in accuracy compared to the initial model. Yet it fails to continue accumulating knowledge and improving the accuracy as training continues. After the third episode, the overall accuracy starts to decay, probably because the knowledge learned from these new episodes interferes with what was learned previously. In contrast, our `Online Continual` learning system (blue) continues to improve its performance and achieves at the end of the 6 episodes an accuracy that matches the accuracy of the model trained with shuffled data under the i.i.d. condition (`Online Joint`, pink). `Offline Joint` (purple) with multiple revisits to the shuffled data achieves the top performance. Note that this is only 8% higher than our continual learning system trained under the online and changing distribution condition.

Figure 8.4 (b) shows the accuracy on all the test data of BB, after each 4 chunks while

learning the 4 episodes. Clearly this series is much harder than BBT. Most of the shots are outdoor and under varied lighting conditions, as also noted in [7]. This corresponds to large distribution changes within and between episodes. Here, `Online` (orange) fails to increase the performance after the first episode. Its accuracy notably fluctuates, probably depending on how (un)related the recently seen data is to the rest of the series. Again, our `Online Continual` learning system (blue) succeeds in improving and accumulating knowledge – up to a 100% improvement over `Online`. Like `Online`, its performance drops at times, yet the drops are dampened significantly, allowing the model to keep on learning over time. Surprisingly, it even outperforms `Online Joint` baseline (pink) and comes close to `Offline Joint` upper bound (purple) that only reaches this accuracy after ten revisits to the training data.

Self Supervision Results

Next we move to the case of learning with self-supervision. This scenario reflects the ideal case where continual learning becomes most interesting. Remember that, as a clue for self-supervision, we use the fact that multiple tracks appearing in the same image should have different identities. We use the six episodes of BBT, although only the first and the sixth episodes actually have a good number of tracks with two persons appearing in one image. Figure 8.5 (a) shows the accuracy on all the episodes after learning each episode. Note how `Online` learning baseline (orange) continues to improve slightly as more episodes are watched. It's only when we get to the last episode, with a larger number of useful tracks, that our `Continual Online` (blue) starts to outperform `Online` baseline.

Figure 8.5 (b) shows the recognition accuracy on the first episode of Mad Men after each chunk. Similar to the previous experiments our `Online Continual` (blue) succeeds in improving the performance and accumulating the knowledge. We conclude that the ability of our system to learn continually is clearly shown, both for weak and self-supervised scenarios.

Ablation Study

Next we perform an ablation study to evaluate the impact of two components of our system. The first factor is the hard buffer used for stabilizing the online training and for updating the importance weights. The second factor is the mechanism for accumulating importance weights across updates. In our system we use a cumulative moving average, which gives all the estimated importance weights the same weight. An alternative is to deploy a decaying average. This reduces the impact of old importance weights in favor of the newest ones. To this end, one can set $\Omega_{\mathcal{T}} = (\Omega_{\mathcal{T}-1} + \Omega^*)/2$ where Ω^* are the currently estimated importance weights. Figure 8.6 shows the accuracy on all the

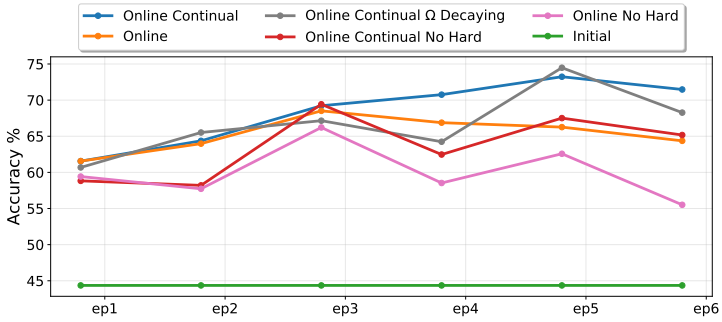


Figure 8.6: A study on the importance of the hard buffer and the cumulative Ω average versus a decaying Ω, figure shows the test accuracy after each episode of BBT.

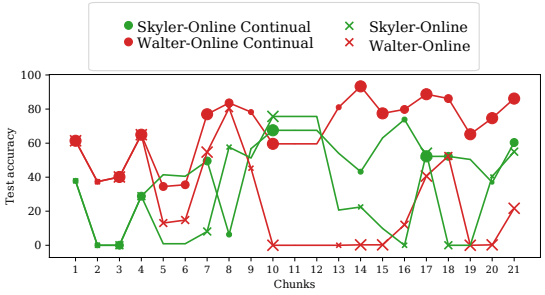


Figure 8.7: A study on the actors recognition during the course of training, figure shows two main actors test accuracy after each chunk in BB.

test data of BBT after each episode achieved by the different variants. The hard buffer clearly improves the performance of both the `Online` and `Online Continual`. The buffer, even if small, gives the learner a chance to re-pass over hard samples and to adjust its gradients for a better estimate of the parameters update step. Additionally, it allows a better estimate of the importance weights used in our `Online Continual` learning system. The decaying average for the importance weights update, leads to more fluctuations due to the higher impact of more recent importance estimates. This allows more forgetting and more bias towards the recent estimate that could be unrepresentative to the overall test data.

Relationship between samples and recognition performance during training: To show how the predictions on the seen actors change over the online training time, we plotted the accuracies per actor after each chunk (for the two most frequent characters of BB, to avoid overloading the figure), see Figure8.7. Marker size indicates the actor's frequency in a chunk, no marker indicates zero appearance. Low frequency in a chunk typically causes the accuracy of `Online` to drop while our method is more stable.

For example the recognition accuracy of "Skyler" drops significantly after learning chunks 13 to 16 for the `Online` baseline. This is due to the very rare appearance of this actor in these chunks. Note that our `Online Continual` maintains relatively high accuracy on recognizing this actor (Skyler) after these chunks. It is also worth mentioning that the performance of `Online` keeps fluctuating during the learning sequence while ours `Online Continual` tend to show more stable behavior after few chunks have been learned. This indicates that the model has accumulated a sufficient knowledge to recognize these actors in the upcoming sequences.

8.4.3 Monocular Collision Avoidance

Collision avoidance is the task in which a robot navigates through the environment aimlessly while avoiding obstacles. We train a neural network to perform this task based on a single input image. Training is done with self-supervision where a simple heuristic based on extra sensors, serves as a supervising expert. The neural network learns to imitate the expert's control, cloning its behavior. Learning to navigate into different environments requires collecting huge amount of training data and as in the case of other machine learning tasks, new environments can always be visited. As such, continually learning to navigate in varying environments using generated supervisory signals is an excellent setup for deploying our system.

Architecture: A neural network model takes a 128x128 RGB frame as input and outputs three discrete steering directions. The architecture consists of 2 convolutional and 2 fully-connected layers with ReLU-activations. Differently from the soap series experiments, training starts from a randomly initialized network. SGD optimizer with cross-entropy loss is used. More details on architecture and hyper-parameters are reported in Table 8.2

Simulation: The experiment is done in a Gazebo simulated environment with the Hector Quadrotor model. The expert is a heuristic reading scans from a Laser Rangefinder mounted on the drone and turning towards the direction with the furthest depth. The demonstration of the expert follows a sequence of four different corridors, referred to as A,B,C and D. The environments differ in texture, obstacles and turns, as shown in Figure 8.8.

Training and Evaluation Every 10 frames a training step occurs, minimizing the training objective. For each model, three networks are trained with different seeds.

We consider the same baselines as in the soap series experiments:`Initial`, `Online`, `Online Joint`, `Offline Joint` in addition to our system `Online Continual`. Note that `Initial` is a randomly initialized model here. Figure 8.9 shows the accuracies achieved during training on the different environments by the compared methods. The accuracy of both `Online` and `Online Continual`

increases in environments where it is currently learning, with grey bars indicating environment changing. However, `Online` tends to forget the early environments while training on new environments. Especially in environment D, the forgetting of earlier environments A and B is outspoken. The cross-entropy loss in environment D rises for all models, indicating a significant change in the data generating distribution.

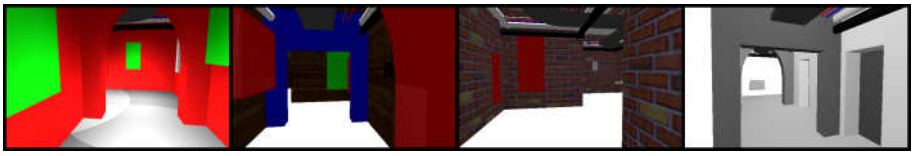


Figure 8.8: Example views in the corridor sequence corresponding to environments A, B, C and D, depicted from left to right.

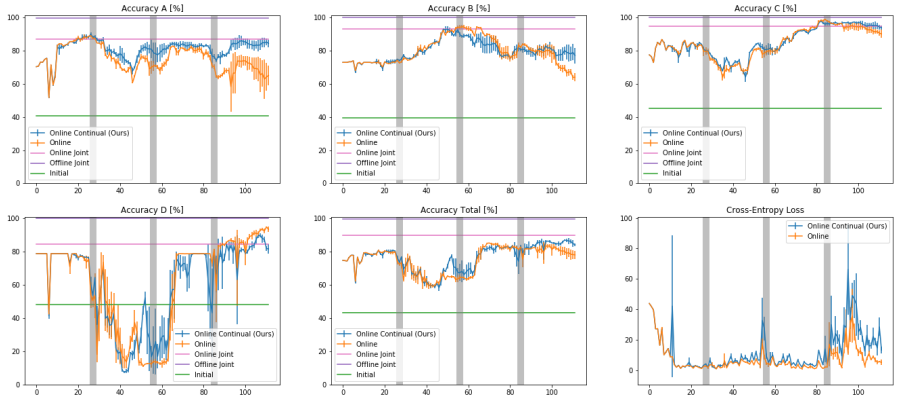


Figure 8.9: Training accuracies on each corridor during learning the (A,B,C,D) sequence. Gray vertical lines indicate the transition to a new environment. The lower right figure shows the cross-entropy loss on the recently received samples. The accuracies of the baselines are added as horizontal lines. Note that the offline joint training accuracy is closer to 100%

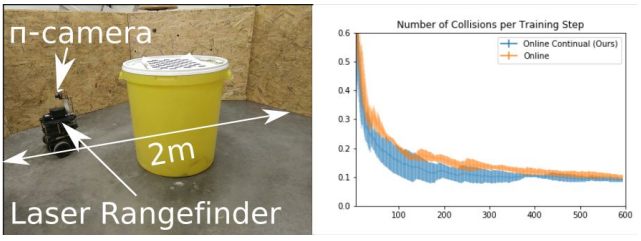


Figure 8.10: Left: Real-world online setup. Right: Number of collisions per training step. Our system speeds up the training, rapidly achieving lower number of collisions.

8.4.4 Proof of Concept in the Real World

In a final experiment, we deploy our online continual learning system on a turtlebot in a small arena in our lab, see Figure 8.10. The model is on-policy pretrained offline in a similar simulated environment. On-policy refers to the model being in control during training instead of the expert. In the previous experiments, our system has proven to be advantageous when experiencing changing environments. In this setup we show that our system is also beneficial during on-policy training within one environment. Again, an expert based on the Laser Rangefinder is providing a self-supervisory signal. On-policy learning tends to be more difficult as the data contains a lot of "dummy" samples when the model visits irrelevant states. This data inefficiency causes the model to learn slower and possibly forget along the way. For example, if the model collides on the left side, the recent signals teach the model to turn right more often. However, after crossing the arena and bumping on the right side, one still wants the model to remember its mistakes made earlier. As such, preserving acquired knowledge over time is crucial for on-policy online learning. In Figure 8.10, we show the number of collisions per step over time with error bars taken over three different models. Clearly our system, `Online Continual`, helps the model to learn faster, with the number of collisions dropping more rapidly than `Online` alone.

8.5 Discussion & Summary

Importance weight regularization appears most effective in online training scenarios when learning environments experience changes while it has a stabilizing effect on the training of one environment. In some cases however, our system tends to slow down the adaptation to newly seen data. Especially when the new data is much more informative or representative than the old, it initially has a negative effect on the training. In other words, pure online learning is faster to adapt to new changes but therefore also inherently less stable. Ultimately, whether the stabilizing effect of continual learning is advantageous or not, depends on the time scale of the changes in the data.

While in this work we focus on a setting where the network architecture remains fixed, and no new outputs or tasks are added over time, in the next chapter we study settings with larger shifts. For instance in a class-incremental setting, an extra output unit could be added to the network each time a new category label appears.

In conclusion, we pushed the limits of current task incremental setting towards online task-free continual learning. We assume an infinite stream of input data, containing changes in the input distribution both gradual and sudden. Our protocol deploys the importance weight regularization in the online setting by deciding when, how and on what data to perform importance weight updates. Its effectiveness is validated

successfully for both supervised and self-supervised learning. More specifically, by using our continual learning method, we demonstrate an improvement of stability and performance over the studied baselines in applications like learning face identities from watching T.V. series and robotic collision avoidance.

Chapter 9

Replay based Online Continual Learning

An ideal continual learning agent learns online with a non-stationary and never-ending stream of data. Regularization based methods tend to suffer from gradual forgetting as no historical samples are re-examined. In general, replay based methods usually store a buffer from the previous data for the purpose of rehearsal. Rehearsal of historical samples allows refreshing old memories in long and largely shifting training sequences. Previous works often depend on task boundary and i.i.d. assumptions to properly select samples for the replay buffer. In this work, aiming for a pure online continual learning, we formulate sample selection as a constraint reduction problem based on the constrained optimization view of continual learning. The goal is to select a fixed subset of constraints that best approximate the feasible region defined by the original constraints. We show that it is equivalent to maximizing the diversity of samples in the replay buffer with parameters gradient as the feature. We further develop a greedy alternative that is cheap and efficient. The advantage of the proposed method is demonstrated by comparing to alternatives under the online continual learning setting. Further comparisons are made against state of the art methods that rely on task boundaries, showing comparable or even better results for our method.

This work was developed during a research stay at Montreal Institute for Learning Algorithms, MILA. This work was published as an article in NeurIPS 2019 [9].

9.1 Introduction

Online continual learning represents a more challenging yet more realistic setting than the task incremental setting. The difficulty comes from the non i.i.d.'ness of the data stream and the unannounced distribution shifts, i.e. task boundaries. Most of the works in the major families of continual learning methods use the relaxed task incremental assumption: the data are streamed one task at a time, with different distributions for each task, while keeping the independent and identically distributed (i.i.d.) assumption and performing offline training within each task. Consequently, they are not directly applicable to the more general setting where data are streamed online with neither i.i.d. assumption nor task boundary information. The parameter isolation family explicitly associates neurons with different tasks, with the consequence that task boundaries are mandatory during both training and testing. Due to the dependency on task boundaries during test, this family of methods tilts more towards multi-task learning than continual learning. Both prior-focused and replay-based methods rely on task boundaries to update the prior or to select a representative set of samples but they have the potential to be adapted to the general setting.

Indeed, in the previous chapter we developed a protocol for deploying our importance weight regularizer, MAS, in the general online continual learning setting. We, however, were aware that a regularization method alone might fail in situations where large distribution shifts are encountered as in incremental classification. This is mainly because the penalty term is a regularizer rather than a hard constraint. The parameter gradually drifts away from the feasible regions of previous tasks, when there is a never ending stream of data. Additionally, the learned decision boundaries evolve during the learning of new classes and optimal parameters for previous classes become outdated. Therefore, it is often necessary to hybridize the prior-focused approach with the replay-based methods for better results [110, 37]. In the previous chapter, we added a small hard buffer to soften the strict online learning scheme but it was rather a very small buffer resembling a short term memory.

The replay-based approach stores the information in the example space either directly in a replay buffer or in a generative model. When learning new data, old examples are reproduced from the replay buffer or generative model, which is used for rehearsal/retraining or used as constraints for the current learning.

In this chapter, we develop strategies to populate the replay buffer under the most general condition where no assumptions are made about the online data stream.

Our contributions are as follows:

- We formulate replay buffer population as a constraint selection problem and formalize it as a solid angle minimization problem.

- We propose a surrogate objective for it and empirically verify that the surrogate objective aligns with the goal of solid angle minimization.
- As a cheap alternative for large sample selection, we propose a greedy algorithm that is as efficient as reservoir sampling yet immune to an imbalanced data stream.
- We compare our method to different selection strategies and show the ability of our solutions to always select a subset of samples that best represents the previous history.
- We perform experiments on continual learning benchmarks and show that our method is on par with, or better than, the previous methods, yet, requiring no i.i.d. assumptions or task boundaries.

In the remainder of this chapter, we discuss the closely related works in Section 9.2. We present the constraint optimization view of continual learning in Section 9.3 and propose our constraint based sample selection method in Section 9.3.2. In Section 9.3.5, we show how to deploy the sample selection in the online setting and present an in-exact greedy alternative for cheap and fast selection. Section 9.4 studies our sample selection methods in different online continual learning scenarios and prove the effectiveness of our approach. Finally, Section 9.5 concludes the chapter.

9.2 Related Work

Recent works that use a replay buffer for continual learning include iCaRL [122] and GEM [92], both of which allocate memory to store a core-set of examples from each task. These methods still require task boundaries in order to divide the storage resource evenly to each task. There are also a few previous works that deal with the situation where task boundary and i.i.d. assumption is not available. For example, reservoir sampling has been employed in [26, 63] so that the data distribution in the replay buffer follows the data distribution that has already been seen. The problem of reservoir sampling is that the minor modes in the distribution with small probability mass may fail to be represented in the replay buffer. As a remedy to this problem, coverage maximization is also proposed in [63]. It intends to keep diverse samples in the replay buffer using Euclidean distance as a difference measure.While the Euclidean distance may be enough for low dimensional data, it could be uninformative when the data lies in a structured manifold embedded in a high dimensional space. In contrast to previous works, we start from the constrained optimization formulation of continual learning, and show that the data selection for the replay buffer is effectively a constraint reduction problem.

9.3 Continual Learning as Constrained Optimization

We consider the supervised learning problem with an online stream of data where one or a few pairs of examples (x, y) are received at a time. The data stream is non-stationary with no assumption on the distribution such as the i.i.d. hypothesis. Our goal is to optimize the loss on the current example(s) without increasing the losses on the previously learned examples.

9.3.1 Problem Formulation

We formulate our goal as the following constrained optimization problem. Without loss of generality, we assume the examples are observed one at a time.

$$\theta^t = \underset{\theta}{\operatorname{argmin}}\, \ell(f(x_t;\theta), y_t) \tag{9.1}$$

$$\text{s.t. } \ell(f(x_i;\theta), y_i) \leq \ell(f(x_i;\theta^{t-1}), y_i);\ \forall i \in [0\,..\,t-1]$$

$f(.;\theta)$ is a model parameterized by θ and ℓ is the loss function. t is the index of the current example and i indexes the previous examples.

As suggested by Gradient episodic memory (GEM) [92], the original constraints can be rephrased to the constraints in the gradient space:

$$\langle g, g_i \rangle = \left\langle \frac{\partial \ell(f(x_t;\theta), y_t)}{\partial \theta}, \frac{\partial \ell(f(x_i;\theta), y_i)}{\partial \theta} \right\rangle \geq 0; \tag{9.2}$$

However, the number of constraints in the above optimization problem increases linearly with the number of previous examples. The required computation and storage resource for an exact solution of the above problem will increase indefinitely with time. It is thus more desirable to solve the above problem approximately with a fixed computation and storage budget. In practice, a *replay buffer* $\mathcal{M}$ limited to M memory slots is often used to keep the previous examples. The constraints are thus only active for $(x_i, y_i) \in \mathcal{M}$. How to populate the replay buffer then becomes a crucial research problem.

GEM [92] assumes access to task boundaries and an i.i.d. distribution within each task episode. It divides the memory budget evenly among the tasks, i.e. $m = M/T$ slots are allocated for each task, where T is the number of tasks. The last m examples from each task are kept in the memory. This has clear limitations when the task boundaries are not available or when the i.i.d. assumption is not satisfied. In this work, we consider

the problem of how to populate the replay buffer in a more general setting where the above assumptions are not met.

9.3.2 Sample Selection as Constraint Reduction

Motivated by Equation 9.1, we set our goal to selecting M examples so that the feasible region formed by the corresponding reduced constraints is close to the feasible region of the original problem. We first convert the original constraints in 9.2 to the corresponding feasible region:

$$C = \bigcap_{i \in [0..t-1]} \{g | \langle g, g_i \rangle \geq 0\} \tag{9.3}$$

Geometrically, C is the intersection of the half spaces described by $\langle g, g_i \rangle \geq 0$, which forms a polyhedral convex cone. The relaxed feasible region corresponding to the replay buffer is:

$$\tilde{C} = \bigcap_{g_i \in \mathcal{M}} \{g | \langle g, g_i \rangle \geq 0\} \tag{9.4}$$

For best approximation of the original feasible region, we require $\tilde{C}$ to be as close to C as possible. It is easy to see that $C \subset \tilde{C}$ because $\mathcal{M} \subset [g_0 \,..\, g_{t-1}]$. We illustrate the relation between C and $\tilde{C}$ in Figure 9.1.

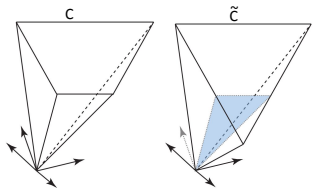


Figure 9.1: Feasible region (polyhedral cone) before and after constraint selection. The selected constraints (excluding the blue one) are chosen to best approximate the original feasible region.

On the left, C is represented while the blue hyperplane on the right corresponds to a constraint that has been removed. Therefore, $\tilde{C}$ (on the right) is larger than C for the inclusion partial order. As we want $\tilde{C}$ to be as close to C as possible, we actually want the "smallest" $\tilde{C}$, where "small" here remains to be defined, as the inclusion order is not a complete order. A good measure of the size of a convex cone is its solid angle defined as the intersection between the cone and the unit sphere.

$$\text{minimize}_{\mathcal{M}} \, \lambda_{d-1} \left(\mathcal{S}_{d-1} \cap \bigcap_{g_i \in \mathcal{M}} \{g | \langle g, g_i \rangle \geq 0\} \right) \tag{9.5}$$

where d denotes the dimension of the space, $\mathcal{S}_{d-1}$ the unit sphere in this space, and λ_{d-1} the Lebesgue measure [1] in dimension $d-1$. Therefore, solving 9.5 would achieve our goal. Note that, in practice, the number of constraints and thus the number of gradients is likely to be way smaller than the dimension of the gradient, which means that the feasible space can be seen as the Cartesian product between its own intersection with $span\,(\mathcal{M})$ and the orthogonal subspace of $span\,(\mathcal{M})$. That being said, we can actually reduce our interest to the size of the solid angle in the M-dimensional space $span\,(\mathcal{M})$, as in 9.6.

$$\text{minimize}_{\mathcal{M}}\ \lambda_{M-1}\left(S_{M-1}^{span(\mathcal{M})} \cap \bigcap_{g_i \in \mathcal{M}} \{g | \langle g, g_i \rangle \geq 0\}\right) \tag{9.6}$$

where $S_{M-1}^{span(\mathcal{M})}$ denotes the unit sphere in $span(\mathcal{M})$. Note that even if the sub-spaces $span(\mathcal{M})$ are different from each other, they all have the same dimension as M, which is fixed, hence comparing their λ_M-measure makes sense. However, this objective is hard to minimize since the formula of the solid angle is complex, as shown in [123] and [15]. Therefore, we propose, in the next section, a surrogate to this objective that is easier to deal with.

9.3.3 An Empirical Surrogate to Feasible Region Minimization

Intuitively, to decrease the feasible set, one must increase the angles between each pair of gradients. Indeed, this is exact in 2D as shown in Figure 9.2. Based on this observation, we propose the surrogate in Equation 9.7.

$$\text{minimize}_{\mathcal{M}} \sum_{i,j \in \mathcal{M}} \frac{\langle g_i, g_j \rangle}{\|g_i\| \|g_j\|} \tag{9.7}$$

$$s.t.\ \mathcal{M} \subset [0\,..\,t-1];\ |\mathcal{M}| = M$$

We empirically studied the relationship between the solid angle and the surrogate function in higher dimensional space using randomly sampled vectors as the gradients.

We faced the problem of no tractable formula for the solid angle, so we estimated it with a sampling method. The estimated angle is an average of Bernoulli random variables with variance then bounded by $\frac{1}{4N}$ where N is the number of these Bernoulli variables. By taking $N = 10^9$, we reach an asymptotic confidence interval of length around 10^{-4}.

[1] Lebesgue measure is a standard way of assigning a measure to subsets of n-dimensional Euclidean space extending the notions of length for intervals, area or volume in higher dimensional space of elementary geometrical sets, such as cubes or rectangles, to more complex sets [156]

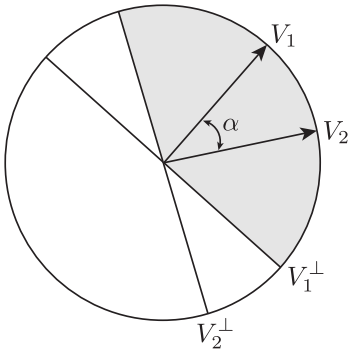


Figure 9.2: Relation between angle formed by two vectors (α) and the associated feasible set (grey region)

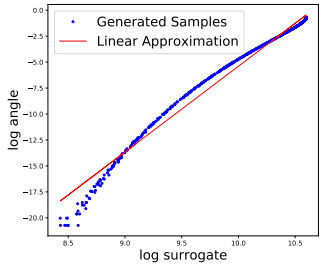


Figure 9.3: Correlation between solid angle and our proposed surrogate in 200D log scale. Note that we only need monotonicity for our objective to hold.

Given a set of sampled vectors, the surrogate value is computed analytically and the solid angle is estimated using Monte Carlo approximation of Equation 9.6. The results are presented in Figure 9.3, which shows a monotonic relation between the solid angle and our surrogate.

9.3.4 Keeping Diverse Samples in the Buffer

Intuitively, keeping diverse samples in the replay buffer could be an efficient way to use the memory budget. It is worth noting that minimizing Equation 9.7 is equivalent to maximizing the variance of the gradient direction as is shown in Equation 9.8.

$$\begin{aligned}\mathrm{Var}_{\mathcal{M}}\left[\frac{g}{\|g\|}\right] =& \frac{1}{M}\sum_{k\in\mathcal{M}}\left\|\frac{g}{\|g\|}\right\|^2 - \left\|\frac{1}{M}\sum_{k\in\mathcal{M}}\frac{g}{\|g\|}\right\|^2 \\ =& 1 - \frac{1}{M^2}\sum_{i,j\in\mathcal{M}}\frac{\langle g_i, g_j\rangle}{\|g_i\|\|g_j\|}\end{aligned} \tag{9.8}$$

This brings up a new interpretation of the surrogate, which is maximizing the diversity of samples in the replay buffer using the parameter gradient as the feature. Note how this is different from maximizing the variance directly on the samples or on the hidden representations, we argue that the parameter gradient could be a better option given its root in Equation 9.1. This is also verified with experiments, Section 9.4.1.

9.3.5 Online Sample Selection

Online sample selection with Integer Quadratic Programming (IQP).

We assume an infinite input stream of data where at each time a new sample(s) is received. From this stream we keep a fixed buffer of size M to be used as a representative of the previous samples. To reduce the computational burden, we use a "recent" buffer in which we store the incoming examples. Once this is full, we perform selection on the union of the replay buffer and the "recent" buffer and replace the samples in the replay buffer with the selection. To perform the selection of M samples, we solve Equation 9.7 as an integer quadratic programming problem.

We first normalize the gradients: $G = \frac{\langle g_i, g_j \rangle}{\|g_i\| \|g_j\|}$ and find a selection vector S that minimizes the following:

$$\begin{aligned} \underset{S}{\text{minimize}} \quad & \frac{1}{2} S^T G S \\ s.t. \quad & \mathbf{1}^T . S = M \\ & s_m \in \{0, 1\} \quad \forall s_m \in S \end{aligned}$$

where $\mathbf{1}$ is a ones vector of the same size as S. Selected samples will correspond to values of 1 in S. The exact procedures of our IQP based online sample selection are described in Algorithm 4. While this is reasonable in the case of a small buffer size, we observed a big overhead when utilizing larger buffers which is likely the case in practical scenarios. The overhead comes from both i) the need to estimate the gradient of each sample in the buffer and the recent buffer and ii) from solving the quadratic problem that is polynomial w.r.t. the size of the buffer. Since this might limit the scalability of our approach, we suggest an alternative greedy method.

An in-exact greedy alternative.

We propose an alternative greedy method based on a heuristic, which could achieve the same goal of keeping diverse examples in the replay buffer, but is much cheaper than performing integer quadratic programming. The key idea is to maintain a score for each sample in the replay buffer. The score is computed by the maximal cosine similarity of the current sample with a fixed number of other random samples in the buffer. When there are two samples similar to each other in the buffer, their scores are more likely to be larger than the others. In the beginning when the buffer is not full, we add incoming samples along with their score to the replay buffer. Once the buffer is full, we randomly select samples from the replay buffer as the candidate to be

replaced. We use the normalized score as the probability of this selection. The score of the candidate is then compared to the score of the new sample to determine whether the replacement should happen or not.

More formally, denote the score as $\mathcal{C}_i$ for sample i in the buffer. Sample i is selected as a candidate to be replaced with probability $P(i) = \mathcal{C}_i / \sum_j \mathcal{C}_j$. The replacement is a Bernoulli event that happens with probability $\mathcal{C}_i/(c + \mathcal{C}_i)$ where $\mathcal{C}_i$ is the score of the candidate and c is the score of the new data. We can apply the same procedure for each example when a batch of new data is received. Algorithm 5 describes the main steps of our gradient based greedy sample selection procedure.

Algorithm 4 IQP Sample Selection

Input: M_r, M_b
function SELECTSAMPLES($\mathcal{M}$, M)
    $\hat{\mathcal{M}} \leftarrow \text{argmin}_{\hat{\mathcal{M}}} \sum_{i,j \in \hat{\mathcal{M}}} \frac{\langle g_i, g_j \rangle}{\|g_i\| \|g_j\|}$
    s.t. $\hat{\mathcal{M}} \subset \mathcal{M}$; $|\hat{\mathcal{M}}| = M$
    return $\hat{\mathcal{M}}$
end function
Initialize: $\mathcal{M}_r$, $\mathcal{M}_b$
Receive: (x, y) ▷ one or few consecutive examples
$\theta \leftarrow$ Update(x, y, $\mathcal{M}_b$; θ)
$\mathcal{M}_r \leftarrow \mathcal{M}_r \cup \{(x, y)\}$
if len($\mathcal{M}_r$) $> M_r$ **then**
    $\mathcal{M}_b \leftarrow \mathcal{M}_b \cup \mathcal{M}_r$
    $\mathcal{M}_r \leftarrow \{\}$
    if len($\mathcal{M}_b$) $> M_b$ **then**
        $\mathcal{M}_b \leftarrow$SelectSamples($\mathcal{M}_b$,M_b)
    end if
end if

Algorithm 5 Greedy Sample Selection

Input: n, M ▷ Number of selected random samples to compare against, buffer size.
Initialize: $\mathcal{M}$, $\mathcal{C}$
Receive: (x, y)
$\theta \leftarrow$Update(x, y, $\mathcal{M}$; θ)
$X, Y \leftarrow$ RandomSubset($\mathcal{M}$, n)
$g \leftarrow \nabla \ell_\theta(x, y)$; $G \leftarrow \nabla_\theta \ell(X, Y)$
$c = \max_i(\frac{\langle g, G_i \rangle}{\|g\| \|G_i\|}) + 1$ ▷ make the score positive
if len($\mathcal{M}$) $>= M$ **then**
    if $c < 1$ **then** ▷ cosine similarity < 0
        $i \sim P(i) = \mathcal{C}_i / \sum_j \mathcal{C}_j$
        $r \sim \text{uniform}(0, 1)$
        if $r < \mathcal{C}_i/(\mathcal{C}_i + c)$ **then**
            $\mathcal{M}_i \leftarrow (x, y)$; $\mathcal{C}_i \leftarrow c$
        end if
    end if
else
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(x, y)\}$; $\mathcal{C} \cup \{c\}$
end if

9.3.6 Constraint vs Regularization

Projecting the gradient of the new sample(s) exactly into the feasible region is computationally very expensive especially when using a large buffer. A usual work around for constrained optimization is to convert the constraints to a soft regularization loss. In our case, this is equivalent to performing rehearsal on the buffer. Note that [25] suggests to constrain only with one random gradient direction from the buffer as a cheap alternative that works equally well to constraining with the gradients of the previous tasks. It was later shown by the same authors [26] that rehearsal on the buffer

achieves a competitive performance. In our method, we do rehearsal while we evaluate both rehearsal and constrained optimization on a small subset of disjoint MNIST and show comparable results.

9.4 Experiments

This section serves to validate our approach and show its effectiveness in dealing with continual learning problems where task boundaries are not available.

Datasets

Disjoint MNIST: MNIST dataset divided into 5 tasks based on the labels with two labels in each task. We use 1k examples per task for training and report results on all test examples.

Permuted MNIST: We perform 10 unique permutations on the pixels of the MNIST images. The permutations result in 10 different tasks with same distributions of labels but different distributions of the input images. Following [92], each of the tasks in permuted MNIST contains only 1k training examples. The test set for this dataset is the union of the MNIST test set with all different permutations.

Disjoint Cifar-10: Similar to disjoint MNIST, the dataset is split into 5 tasks according to the labels, with two labels in each task. As this is harder than MNIST, we use a total of 10k training examples with 2k examples per task.

In all experiments, we use a fixed batch size of 10 samples and perform few iterations over a batch (1-5). Note that this is different from multiple epochs over the whole data. For disjoint MNIST, we report results using different buffer sizes in table 9.1. For permuted MNIST results are reported using buffer size 300 while for disjoint Cifar-10 we couldn't get sensible performance for the studied methods with buffer size smaller than 1k. All results are averaged over 3 different random seeds.

Models

Following [92], for disjoint and permuted MNIST we use a two-layer neural network with 100 neurons each while for Cifar-10 we use ResNet18. Note that we employ a shared head in the incremental classification experiments, which is much more challenging than the multi-head used in [92].

9.4.1 Comparison with Sample Selection Baselines

We want to study the buffer population in the context of the online continual learning setting when no task information is present and no assumption on the data generating distribution is made. Since most existing works assume knowledge of task boundaries, we decide to deploy 3 baselines along with our two proposed methods. Given a fixed buffer size M we compare the following:

Random (`Rand`): Whenever a new batch is received, it joins the buffer. When the buffer is full, we randomly select samples to keep size M from the new batch and samples already in buffer.

Online Clustering: A possible way to keep diverse samples in the buffer is online clustering with the goal of selecting a set of M centroids. This can be done either in the feature space or in the gradient space. In the feature space we consider a selection baseline, denoted as Feature based Sample Selection (`FSS-Clust`), where we use as a metric the distance between the samples features, here the last layer before classification. For the selection baseline in the gradient space, denoted as Gradient based Sample Selection (`GSS-Clust`), we consider as a metric the Euclidean distance between the normalized gradients. We adapted the doubling algorithm for incremental clustering described in [23].

IQP Gradient based Sample Selection (`GSS-IQP`**)**: Our surrogate to select samples that minimize the feasible region described in Equation 9.7 and solved as an integer quadratic programming problem. Due to the cost of computation we report our `GSS-IQP` on permuted MNIST and disjoint MNIST only.

Gradient based Greedy Sample Selection (`GSS-Greedy`**)**: Our greedy selection variant detailed in Algorithm 5. Note that differently from previous selection strategies, it doesn't require re-processing all the recent and buffer samples to perform the selection which is a huge gain in the online streaming setting.

Method \ Buffer Size	300	400	500
`Rand`	37.5	45.9	57.9
`GSS-IQP(ours)`	75.9	82.1	84.1
`GSS-Clust`	75.7	81.4	83.9
`FSS-Clust`	75.8	80.6	83.4
`GSS-Greedy(ours)`	**82.6**	**84.6**	**84.8**

Table 9.1: Average test accuracy in % of sample selection methods on disjoint MNIST with different buffer sizes.

Method	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Avg.
Rand	66.29	58.82	67.9	59.93	71.45	71.31	79.57	80.3	84.3	**86.9**	72.7
GSS-IQP(ours)	74.1	69.73	70.77	70.5	73.34	78.6	81.8	81.8	**86.4**	85.45	77.3
GSS-Clust	75.4	**76.66**	**77.89**	**73.56**	**80.7**	**80.83**	**82.4**	**83.53**	83.8	84.75	**79.96**
FSS-Clust	82.2	71.34	76.9	70.5	70.56	74.9	77.68	79.56	82.7	85.3	77.8
GSS-Greedy(ours)	**83.35**	70.84	72.48	70.5	72.8	73.75	79.86	80.45	82.56	84.8	77.3

Table 9.2: Comparison of different selection strategies on permuted MNIST benchmark, buffer size 300. Table shows test accuracies in % on each task at the end of the training sequence.

Method	T1	T2	T3	T4	T5	Avg.
Rand	0	0.49	5.68	**52.18**	84.96	28.6
GSS-Clust	0	5.91	15.91	12.62	78.14	22.5
FSS-Clust	0.17	0.82	5.42	38.12	**87.90**	26.7
GSS-Greedy(ours)	**42.36**	**14.61**	**13.60**	19.30	77.83	**33.56**

Table 9.3: Comparison of different selection strategies on disjoint Cifar-10 benchmark, buffer size 1k. Table shows test accuracies in % on each task at the end of the training sequence.

9.4.2 Performance of Sample Selection Methods

Tables 9.1, 9.2, 9.3 report the test accuracy on each task at the end of the data stream for disjoint MNIST, permuted MNIST, disjoint Cifar-10 respectively. First of all, the accuracies reported in the tables might appear lower than state of the art numbers. This is due to the strict online setting, the use of a shared head and more importantly the no use of task boundary. In contrast, all previous works assume availability of task boundary either at training or both at training and testing. The performance of the random baseline Rand clearly indicates the difficulty of this setting. It can be seen that both of our selection methods exhibit stable and good performance over the different buffer sizes on different benchmarks. Notably, the gradient based clustering GSS-Clust performs comparably and even favorably on permuted MNIST to the feature clustering FSS-Clust suggesting the effectiveness of a gradient based metric in the continual learning setting. Surprisingly, GSS-Greedy performs on par and even better than the other selection strategies especially on disjoint Cifar-10 indicating not only a cheap but a strong sample selection strategy.

9.4.3 Performance under Blurry Task Boundary

An interesting setting is the scenario where there are no clear task boundaries in the data stream. As we mentioned in the introduction, such situation can happen in practice.

We start by blurring the task boundaries in disjoint Cifar-10 benchmark. For each task in the dataset, we keep the majority of the examples while we randomly swap a small percentage of the examples with other tasks. A larger swap percentage corresponds to more blurry boundaries. This is a similar setting to what we used in Chapter 7, Section 7.4.6. We keep 90% of the data for each task, and introduce 10% of data from the other tasks. We make comparisons to the other studied selection methods. Since tasks are not disjoint, forgetting is not as severe as in the case of complete disjoint tasks. Hence, we use a buffer of 500 samples and train on 1k samples per task which allows us to run our `GSS-IQP` more smoothly.

Table 9.4 reports the accuracy of each task at the end of the sequence. Both of our methods perform better than other selection strategies.

Method	T1	T2	T3	T4	T5	Avg.
`Rand`	0	3.45	9.85	**54.67**	78.76	29.0
`GSS-IQP(ours)`	9.38	**11.33**	**17.05**	30.84	**79.53**	**29.6**
`GSS-Clust`	2.43	16.75	9.09	20.71	77.98	25.0
`FSS-Clust`	2.95	05.09	6.06	38.16	78.14	26.0
`GSS-Greedy(ours)`	**34.2**	11.14	14.96	20.25	67.5	**29.6**

Table 9.4: Comparison of different selection strategies on disjoint Cifar-10 with blurry task boundary, buffer size 500. Table shows test accuracies in $\%$ on each task at the end of the training sequence.

9.4.4 Constrained Optimization Compared to Rehearsal

By the end of Section 9.3.5, we have elaborated on the computational complexity of the constrained optimization, which renders then infeasible with large buffers. That's mainly because at each learning step, gradient of each sample in the buffer needs to be estimated and then the new sample gradient needs to be projected onto the feasible region determined by all the samples gradients. As an alternative, we perform rehearsal on the buffer. Here, we want to compare the performance of the two update strategies, the constrained optimization `GSS-IQP(Constrained)` and the rehearsal `GSS-IQP(Rehearsal)`. We consider disjoint MNIST benchmark and use 200 training samples per task. Table 9.5 reports the test accuracy on each task achieved by each strategy at the end of the training when using a buffer of size 100 while table 9.6 reports the accuracy with 200 buffer size. `GSS-IQP(Constrained)` improves over `GSS-IQP(Rehearsal)` with a margin of $3-5\%$ but requires a long time to train as it scales polynomialy with the number of samples in the buffer apart from the need to compute the gradient of each buffer sample at each training step. `GSS-IQP(Rehearsal)` with larger buffer is less computational and yields similar

Method	T1	T2	T3	T4	T5	Avg.
GSS-IQP(Constrained)	90.0	70.0	45.13	88.77	86.08	76.26
GSS-IQP(Rehearsal)	81.5	69.47	46.96	69.80	88.0	71.3

Table 9.5: Comparison between Rehearsal and Constrained optimization with our GSS-IQP method on disjoint MNIST and buffer size 100. Table shows test accuracies in $\%$ on each task at the end of the training sequence.

Method	T1	T2	T3	T4	T5	Avg.
GSS-IQP(Constrained)	95.0	83.0	68.7	87.6	82.4	83.4
GSS-IQP(Rehearsal)	94.6	83.89	50.6	77.0	88.67	78.9

Table 9.6: Comparison between Rehearsal and Constrained optimization with our GSS-IQP method on disjoint MNIST and buffer size 200. Table shows test accuracies in $\%$ on each task at the end of the training sequence.

results, comparing GSS-IQP(Rehearsal) with a buffer of size 200 (78.9%) and GSS-IQP(Constrained) with a buffer of size 100 (76.26%).

9.4.5 Comparison with Reservoir Sampling

Reservoir sampling [155] is a simple replacement strategy to fill the memory buffer when the task boundaries are unknown based on the underlying assumption that the overall data stream is i.i.d. distributed. It would work well when each of the tasks has a similar number of examples. However, it could lose the information on the under-represented tasks if some of the tasks have significantly fewer examples than the others. In this chapter, we study and propose algorithms to sample from an imbalanced stream of data. Our strategy has no assumption on the data stream distribution, hence it could be less affected by imbalanced data, which is often encountered in practice.

We test this scenario on disjoint MNIST. We modify the data stream to settings where one of the tasks has an order of magnitude more samples than the rest, generating 5 different sequences where the first sequence has 2000 samples of the first task and 200 from each other task, the second sequence has 2000 samples from the second task and 200 from others, and same strategy applies to the rest of the sequences. Table 9.7 reports the average accuracy at the end of each sequence over 3 runs with 300 samples as buffer size. It can be clearly seen that our selection strategies outperform reservoir sampling. Our improvement reaches 15%. While reservoir sampling has been introduced by [25] as a cheap powerful strategy to populate the buffer samples, here we show that the effectiveness of such strategy cannot be taken for granted.

Method	Seq1	Seq2	Seq3	Seq4	Seq5	Avg.
GSS-IQP(ours)	**75.9**	76.2	79.06	76.6	74.7	76.49
GSS-Greedy(ours)	71.2	**78.5**	**81.5**	**79.5**	**79.1**	**77.96**
Reservoir	63.7	69.4	66.8	69.1	76.6	69.12

Table 9.7: Comparison with reservoir sampling on different imbalanced data sequences from disjoint MNIST, buffer size 300. Table shows average test accuracies in % on each sequence achieved at its end.

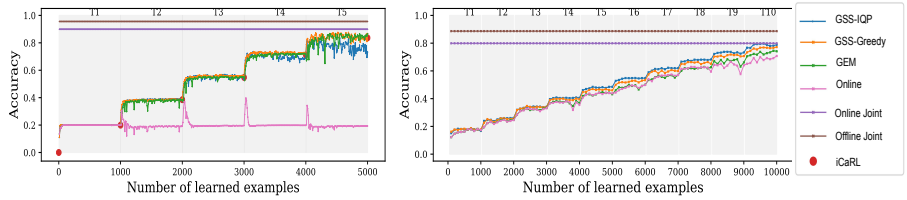
Having shown the robustness of our selection strategies both GSS−IQP and GSS−Greedy, we move now to compare with state of the art replay-based methods that allocate a separate buffer per task and only play samples from previous tasks during the learning of others.

9.4.6 Comparison with State of the Art Task Aware Methods

Our method ignores any tasks information which places us at a disadvantage because the methods that we compare to utilize the task boundaries as an extra information. In spite of this disadvantage, we show that our method performs similarly on these datasets. We consider the following methods:

Online is a model trained online on the stream of data without any mechanism to prevent forgetting.
Online Joint is the Online baseline trained on an i.i.d. stream of the data.
Offline Joint is a model trained offline for multiple epochs with i.i.d. sampled batches and serves as an upper bond.
GEM [92] stores a fixed amount of random examples per task and uses them to provide constraints when learning new examples.
iCaRL [122] follows an incremental classification setting. It also stores a fixed number of examples per class but uses them to rehearse the network when learning new information.

For ours, we report both GSS−IQP and GSS−Greedy on permuted and disjoint MNIST and only GSS−Greedy on disjoint Cifar-10 due to the computational burden. Since we perform multiple iterations over a given batch still in the online setting, we treat the number of iterations as a hyper parameter for GEM and iCaRL. We found that GEM performance constantly deteriorates with multiple iterations while iCaRL improves. Figure 9.4a shows the test accuracy on disjoint MNIST which is evaluated during the training procedure at an interval of 100 training examples with 300 buffer size. For the i.i.d. baselines, we only show the achieved performance at the end of the training. For iCaRL, we only show the accuracy at the end of each task because iCaRL uses the selected exemplars for prediction that only happens at the end of each



(a) Disjoint MNIST (b) Permuted MNIST

Figure 9.4: Comparison with state of the art task aware replay methods on disjoint MNIST and permuted MNIST, buffer size 300. Figures show test accuracy in %.

task. We observe that both variants of our method have a very similar learning curve to GEM except the few last iterations where `GSS-IQP` performance slightly drops.

Figure 9.4b compares our methods with the baselines and `GEM` on the permuted MNIST dataset. Note that `iCaRL` is not included, as it is designed only for incremental classification. From the good performance of the `Online` baseline it is apparent that permuted MNIST has less interference between the different tasks. Our two variants perform better than GEM and get close to `Online Joint` performance.

Figure 9.5 shows the accuracy on disjoint Cifar-10 evaluated during the training procedure at an interval of 100 training examples. `GSS_Greedy` shows better performance than `GEM` and `iCaRL`, and it even achieves a better average test performance at the end of the sequence than `Online Joint`. We found that `GEM` suffers more forgetting on previous tasks while `iCaRL` shows lower performance on the last task. Note that our setting is much harder than offline tasks training used in iCaRL [122].

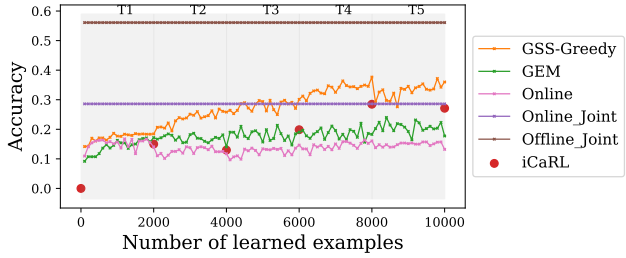


Figure 9.5: Comparison with state of the art task aware replay methods on disjoint Cifar-10, buffer size 1k. Figure show test accuracy in %.

9.5 Summary

In this chapter, we studied online continual learning on a never ending stream of data sampled from largely shifting distribution as in the case of incremental classification with no task boundaries and shared output layer. We prove that in the online continual learning setting we can smartly select a finite number of samples to be representative of all previously seen data without knowing task boundaries. We aim for samples diversity in the gradient space and introduce a greedy selection approach that is efficient and constantly outperforming other selection strategies. We still perform as well as algorithms that use the knowledge of task boundaries to select the representative examples. Moreover, our selection strategy gives us advantage under the settings where the task boundaries are blurry or data are imbalanced.

Chapter 10

Conclusion

In this chapter, we first summarize the main contributions of the work done during the course of this PhD. We then discuss the advances within the continual learning field in the past few years while shedding light on the impact and limitations of the methods proposed in this thesis. We conclude this thesis by pointing at some promising research directions towards realizing continual learning.

10.1 Summary of Contributions

While the catastrophic forgetting problem has been studied since the early development of neural networks, continual learning is an emerging field that only recently started to have a well acknowledged definition. During the first year of my PhD, I was working on the topic of domain adaptation and had presented two methods, not described in this manuscript. The first one proposed an on the fly domain adaptation method [11] and the other was more of a practical domain adaptation method for actor recognition in T.V. series [7]. Domain adaptation is limited to two tasks/domains and unidirectional. After a period of extensive research, we started working on the concept of incremental experts learning on different domains extending transfer learning to an unlimited sequence of domains. The idea evolved to the work described in Chapter 4. By designing a sequentially learned gate, we showed expert level performance on each domain maintaining a deployment time close to that of one model, without assuming any access to data from previous domains. Our work, Expert Gate, was among the first works to propose a solution for inferring the task ID at test time given unseen data. This has a great advantage as it overcomes the need for an extra dependence on human intervention in the continual learning cycle. Most of the subsequent works on avoiding

catastrophic forgetting and on continual learning require an imaginary task oracle at test time. It is worth mentioning that Expert Gate was successfully deployed in industry on cases with more than 80 domains learned in a sequence. During the last development stages of Expert Gate, a closely related method [134] was proposed by a team from Deep Mind. It grows a neural network by connecting a new module for each new task. Concurrently, Learning without Forgetting [85] (LwF) proposed to deploy the knowledge distillation loss for learning a sequence of tasks using one shared neural network model. In Expert Gate, we have shown that LwF is vulnerable to forgetting when there is a significant distribution shift between tasks but beneficial when related tasks are being learned. As such we have implemented LwF as a method for knowledge transfer between domains.

Expert Gate provides expert level performance at the expense of storing all the previously learned models. However, as explained in Chapter 5, many scenarios emerge where there is a need to learn the different upcoming tasks using a single model. Starting from the same concept of sequentially learning light shallow autoencoders, we have proposed a solution for incremental task learning using one shared model. The autoencoders are used here to preserve the important features learned for the previous tasks during the learning of other tasks. In Chapter 5 we have presented our Encoder Based Lifelong Learning work (EBLL) and show that it is advantageous over LwF mostly when the learned tasks are not largely related. LwF and EBLL use the data of the new task as a proxy for the previous tasks data, hence they are sensitive to big distribution shifts.

Another line of methods had emerged [69, 168]: prior-focused methods that use the knowledge acquired from the previous tasks as a prior when learning the new tasks. An ℓ_2 regularization is used to penalize the changes on the important parameters. These methods are problem agnostic and can have a constant memory consumption [168] which are very desired criteria in continual learning. However, the important parameters are estimated on all the training data of a given task and then remained fixed for the rest of an agent lifetime, continual learning methods must be adaptive. In Chapter 6, we have argued that given an unlimited sequence of tasks and a fixed model capacity; it is impossible to preserve all the previous knowledge while still being able to learn new tasks. Instead of gradually forgetting equally all the previous knowledge, we proposed to learn the important bits of previous knowledge, those that are frequently used at deployment time. We further prove that Hebbian learning can be seen as a local version of our method. We have shown the ability of our method, MAS, to learn what is important not to forget based on a specific test setting using any existing data. This brings into realization two additional desired characteristics of continual learning, adaptive and graceful forgetting. Our method also showed at the time of development, state of the art performance on the standard task incremental setting.

At the time of developing MAS, our aim was to move the task incremental setting towards a smoother continual learning. We thus proposed a new benchmark for

continual learning. Instead of learning tasks as different as birds and digits, we proposed to learn facts, <Subject, Object, Predicate> triplets. This eliminates the multi-head setting that hides most of the difficulties of continual learning, since each task has a separate unshared head. This work was published as a conference paper at ACCV2018 [34] but not detailed in this manuscript.

Overcoming catastrophic forgetting in the absence of the previous tasks data and given a fixed model capacity, relies mostly on identifying the important features/parameters for a previous task and preserving them during the learning process. As we showed in Chapter 7, an important aspect is that the optimization process of a given task should be aware of future upcoming tasks to be learned. This plays the role of not utilizing all the model capacity but rather accounting for the future tasks. We have shed the light on the importance of sparsity in continual learning and studied different regularization criteria and activation functions in the context of continual learning. Inspired by neural inhibition in the neural biology, we proposed a sparsity imposing regularizer that shows larger rates of sparsity compared to other regularizers. Our regularizer improved significantly the continual learning performance on different studied datasets.

All the previous works, including ours, focus on the milder task incremental setting that maintains the offline training of each task and relies on knowing the task boundaries to process the acquired knowledge. While catastrophic forgetting seemed to be less of a problem given the new advances, many of the real applications seemed far fetched under the task incremental setting. In Chapter 8, we proposed a protocol for deploying our importance weight regularizer, MAS, in the online continual learning setting, removing any dependence on the task boundaries and performing online training on streaming data. We have shown improved learning behaviour and less forgetting when learning from streaming data in gradually changing environments such as recognizing actors in TV series or learning to avoid obstacles.

Online learning from a non stationary distribution is particularly challenging. The inferred decision boundaries will change during the learning process and previously acquired knowledge might soon become outdated as new information is processed. Solving continual learning without storing any sample from the previous history seems extremely hard in settings with largely shifting distributions. In the last chapter (9) we have studied online continual learning using a buffer of historical samples from the previously seen data. We formulate online continual learning as a constrained optimization problem and propose a gradient based sample selection procedure. In spite of not relying on any task information for selecting the stored samples, we have shown improved performance on different continual learning scenarios.

Our works on online continual learning were the first to explore the challenging yet more realistic online continual learning setting. Only very recently new research works have started to explore this setting [54, 91].

The samples stored from the previous history are usually replayed randomly while learning new samples. While not explained in this manuscript, we have lately studied replay strategies during online learning and propose a method that searches for what samples would be the most interfered given an estimated parameters update. This work was joint with researchers from MILA following my research visit. Its outcome was accepted as an article in NEURIPS 2019 (Conference on Neural Information Processing Systems).

Finally, during the past last year, I have worked with colleges from our group on a continual learning survey. We have proposed two benchmarks for continual learning and evaluated the different existing families of continual learning methods. This was done while studying various effects such as tasks ordering and regularization. This survey is under review with the IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) journal.

10.2 Discussion and Future Research Directions

In the introduction, Chapter 1, we listed 10 desired characteristics of continual learning, namely, constant memory, no task boundaries, online learning, forward transfer, backward transfer, problem agnostic, adaptive, no test time oracle, task revisiting and graceful forgetting. This list of characteristics has evolved during the course of this PhD work. While the main target is to solve the stability/plasticity dilemma, meeting some if not all the desired characteristics becomes crucial for the realization of a continual learning system.

In Chapter 4, we tolerated a linear increase in memory in order to obtain expert level performance on all tasks. There seems to be a better compromise that can be achieved. For example, instead of learning a model per task, the initial model can be divided into various modules, each specialized in a different task. A possible solution is to identify the important parameters for a given problem and isolate them when learning to solve different problems. The sequentially learned gates would be deployed to forward the data through the model in a similar manner to [138]. Here, we intentionally state that specialization should apply to a “problem” indicating a group of similar tasks. Within each of these modules, identified for a given problem, consolidation mechanisms as what we proposed in Chapter 6 can be imposed to ensure that the newly encoded knowledge doesn’t catastrophically interfere with what has previously been acquired. While catastrophic forgetting must be overcome, graceful forgetting should be allowed to insure the adaptivity of the system.

It seems that complex biological systems have hierarchical organization [102], or the so called modules within modules suggesting that neural modular composition might be necessary for achieving real continual learning of different tasks and problems. An

attractive approach that can be coupled with continual learning is meta learning. A possible direction is that besides the parameters being trained on the received tasks, a meta model would control the modularization of the information to reduce the interference and increase the sharing. This is trained via a meta objective optimizing the forward/backward transfer gained during the training cycle.

Another important aspect is the deployment of memory in a continual learning system. Achieving online continual learning involves continuous and fast adaptation in addition to a change of previous beliefs. This can't be realized without replaying previous memories. In fact, there seems to be evidence for experience replay between hippocampus and neocortex in order to accommodate the recent events while preserving the old memories [59]. This can be achieved in continual learning by relying on a buffer sampled from the previous history as we proposed in Chapter 9 or via generative modelling. A generator can be trained to capture the data generating distribution and acts as connectionist memory [118, 81]. However, generative modelling is extremely hard to train online especially on a changing data distribution. This leaves big room for future research.

Overall, we believe that the contributions of this PhD target different aspects of continual learning and further investigation on how to combine the proposed techniques is necessary to move a step forward towards solving the continual learning problem.

Bibliography

[1] A OLSHAUSEN, B., AND FIELD, D. Sparse coding of sensory inputs. *Current opinion in neurobiology 14* (09 2004), 481–7.

[2] AGHASI, A., ABDI, A., NGUYEN, N., AND ROMBERG, J. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems* (2017), pp. 3180–3189.

[3] AHMED, K., BAIG, M. H., AND TORRESANI, L. Network of experts for large-scale image categorization. In *Computer Vision – ECCV 2016* (Cham, 2016), B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Springer International Publishing, pp. 516–532.

[4] ALAIN, G., AND BENGIO, Y. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research 15*, 1 (2014), 3563–3593.

[5] ALJUNDI, R., BABILONI, F., ELHOSEINY, M., ROHRBACH, M., AND TUYTELAARS, T. Memory aware synapses: Learning what (not) to forget. In *European Conference on Computer Vision* (2018), Springer, pp. 144–161.

[6] ALJUNDI, R., CHAKRAVARTY, P., AND TUYTELAARS, T. Expert gate: Lifelong learning with a network of experts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[7] ALJUNDI, R., CHAKRAVARTY, P., AND TUYTELAARS, T. Who's that actor? automatic labelling of actors in tv series starting from imdb images. In *Asian Conference on Computer Vision* (2016), Springer, pp. 467–483.

[8] ALJUNDI, R., KELCHTERMANS, K., AND TUYTELAARS, T. Task-free continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 11254–11263.

[9] ALJUNDI, R., LIN, M., GOUJAUD, B., AND BENGIO, Y. Online continual learning with no task boundaries. *CoRR abs/1903.08671* (2019).

[10] ALJUNDI, R., ROHRBACH, M., AND TUYTELAARS, T. Selfless sequential learning. In *International Conference on Learning Representations* (2019).

[11] ALJUNDI, R., AND TUYTELAARS, T. Lightweight unsupervised domain adaptation by convolutional filter reconstruction. In *European Conference on Computer Vision* (2016), Springer, pp. 508–515.

[12] ANS, B., AND ROUSSET, S. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'Académie des Sciences-Series III-Sciences de la Vie 320*, 12 (1997), 989–997.

[13] ATKINSON, C., MCCANE, B., SZYMANSKI, L., AND ROBINS, A. V. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. *CoRR abs/1802.03875* (2018).

[14] BALNTAS, V., RIBA, E., PONSA, D., AND MIKOLAJCZYK, K. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC* (2016), vol. 1, p. 3.

[15] BECK, M., ROBINS, S., AND SAM, S. V. Positivity theorems for solid-angle polynomials. *arXiv preprint arXiv:0906.4031* (2009).

[16] BENDALE, A., AND BOULT, T. Towards open world recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1893–1902.

[17] BENGIO, Y., ET AL. Learning deep architectures for ai. *Foundations and trends® in Machine Learning 2*, 1 (2009), 1–127.

[18] BENGIO, Y., AND LECUN, Y., Eds. *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (2014).

[19] BOTTOU, L. Online learning and stochastic approximations. *On-line learning in neural networks 17*, 9 (1998), 142.

[20] BOTTOU, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[21] BOURLARD, H., AND KAMP, Y. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics 59*, 4-5 (1988), 291–294.

[22] CARUANA, R. Multitask learning. In *Learning to learn*. Springer, 1998, pp. 95–133.

[23] CHARIKAR, M., CHEKURI, C., FEDER, T., AND MOTWANI, R. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing 33*, 6 (2004), 1417–1440.

[24] CHAUDHRY, A., DOKANIA, P. K., AJANTHAN, T., AND TORR, P. H. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 532–547.

[25] CHAUDHRY, A., RANZATO, M., ROHRBACH, M., AND ELHOSEINY, M. Efficient lifelong learning with a-GEM. In *International Conference on Learning Representations* (2019).

[26] CHAUDHRY, A., ROHRBACH, M., ELHOSEINY, M., AJANTHAN, T., DOKANIA, P. K., TORR, P. H., AND RANZATO, M. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486* (2019).

[27] CHEN, Z., AND LIU, B. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning 12*, 3 (2018), 1–207.

[28] COGSWELL, M., AHMED, F., GIRSHICK, R., ZITNICK, L., AND BATRA, D. Reducing overfitting in deep networks by decorrelating representations. *arXiv preprint arXiv:1511.06068* (2015).

[29] CORDTS, M., OMRAN, M., RAMOS, S., REHFELD, T., ENZWEILER, M., BENENSON, R., FRANKE, U., ROTH, S., AND SCHIELE, B. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[30] CSURKA, G. *Domain adaptation in computer vision applications*. Springer, 2017.

[31] DE CAMPOS, T. E., BABU, B. R., AND VARMA, M. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal* (February 2009).

[32] DONAHUE, J., JIA, Y., VINYALS, O., HOFFMAN, J., ZHANG, N., TZENG, E., AND DARRELL, T. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML* (2014), pp. 647–655.

[33] DUCHI, J., HAZAN, E., AND SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12*, Jul (2011), 2121–2159.

[34] ELHOSEINY, M., BABILONI, F., ALJUNDI, R., ROHRBACH, M., AND TUYTELAARS, T. Exploring the challenges towards lifelong fact learning. In *Asian Conference on Computer Vision* (2018).

[35] ELHOSEINY, M., COHEN, S., CHANG, W., PRICE, B. L., AND ELGAMMAL, A. M. Sherlock: Scalable fact learning in images. In *AAAI* (2017), pp. 4016–4024.

[36] EVERINGHAM, M., VAN GOOL, L., WILLIAMS, C. K. I., WINN, J., AND ZISSERMAN, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[37] FARQUHAR, S., AND GAL, Y. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733* (2018).

[38] FERNANDO, C., BANARSE, D., BLUNDELL, C., ZWOLS, Y., HA, D., RUSU, A. A., PRITZEL, A., AND WIERSTRA, D. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734* (2017).

[39] FRENCH, R. M. Semi-distributed representations and catastrophic forgetting in connectionist networks. *Connection Science 4*, 3-4 (1992), 365–377.

[40] FRENCH, R. M. Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. *network 1111* (1994), 00001.

[41] FRENCH, R. M. Pseudo-recurrent connectionist networks: An approach to the 'sensitivity-stability' dilemma. *Connection Science 9*, 4 (1997), 353–380.

[42] FRENCH, R. M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences 3*, 4 (1999), 128–135.

[43] FRENCH, R. M., AND FERRARA, A. Modeling time perception in rats: Evidence for catastrophic interference in animal learning. In *Proceedings of the 21st Annual Conference of the Cognitive Science Conference* (1999), Citeseer, pp. 173–178.

[44] GEIGER, A., LENZ, P., STILLER, C., AND URTASUN, R. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research* (2013), 0278364913491297.

[45] GIRSHICK, R., DONAHUE, J., DARRELL, T., AND MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2014), pp. 580–587.

[46] GLOROT, X., BORDES, A., AND BENGIO, Y. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (Fort Lauderdale, FL, USA, 11–13 Apr

2011), G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15 of *Proceedings of Machine Learning Research*, PMLR, pp. 315–323.

[47] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. Deep learning. Book in preparation for MIT Press, 2016.

[48] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680.

[49] GOODFELLOW, I. J., MIRZA, M., XIAO, D., COURVILLE, A., AND BENGIO, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211* (2013).

[50] GOODFELLOW, I. J., WARDE-FARLEY, D., MIRZA, M., COURVILLE, A., AND BENGIO, Y. Maxout networks. *arXiv preprint arXiv:1302.4389* (2013).

[51] GROSSBERG, S. *Studies of mind and brain : neural principles of learning, perception, development, cognition, and motor control*. Boston studies in the philosophy of science 70. Reidel, Dordrecht, 1982.

[52] HAMMER, B., ET AL. Incremental learning algorithms and applications. In *ESANN* (2016).

[53] HAZAN, E., RAKHLIN, A., AND BARTLETT, P. L. Adaptive online gradient descent. In *Advances in Neural Information Processing Systems* (2008), pp. 65–72.

[54] HE, X., SYGNOWSKI, J., GALASHOV, A., RUSU, A. A., TEH, Y. W., AND PASCANU, R. Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201* (2019).

[55] HEBB, D. The organization of behavior. 1949. *New York Wiely* (2002).

[56] HESSEL, M., MODAYIL, J., VAN HASSELT, H., SCHAUL, T., OSTROVSKI, G., DABNEY, W., HORGAN, D., PIOT, B., AZAR, M., AND SILVER, D. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298* (2017).

[57] HINTON, G., VINYALS, O., AND DEAN, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop* (2015).

[58] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[59] HONEY, C. J., NEWMAN, E. L., AND SCHAPIRO, A. C. Switching between internal and external modes: a multiscale learning principle. *Network Neuroscience 1*, 4 (2017), 339–356.

[60] HSU, Y.-C., LIU, Y.-C., AND KIRA, Z. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488* (2018).

[61] HU, T. Online regression with varying gaussians and non-identical distributions. *Analysis and Applications 9*, 04 (2011), 395–408.

[62] HUSZÁR, F. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences 115*, 11 (2018), E2496–E2497.

[63] ISELE, D., AND COSGUN, A. Selective experience replay for lifelong learning. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[64] JACOB, L., VERT, J.-P., AND BACH, F. R. Clustered multi-task learning: A convex formulation. In *Advances in neural information processing systems* (2009), pp. 745–752.

[65] JACOBS, R. A., JORDAN, M. I., NOWLAN, S. J., AND HINTON, G. E. Adaptive mixtures of local experts. *Neural computation 3*, 1 (1991), 79–87.

[66] JAVED, K., AND WHITE, M. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588* (2019).

[67] JIA, X., DE BRABANDERE, B., TUYTELAARS, T., AND GOOL, L. V. Dynamic filter networks. In *Advances in Neural Information Processing Systems* (2016), pp. 667–675.

[68] JUNG, H., JU, J., JUNG, M., AND KIM, J. Less-forgetting Learning in Deep Neural Networks. Tech. rep., 2016.

[69] KIRKPATRICK, J., PASCANU, R., RABINOWITZ, N., VENESS, J., DESJARDINS, G., RUSU, A. A., MILAN, K., QUAN, J., RAMALHO, T., GRABSKA-BARWINSKA, A., ET AL. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* (2017), 201611835.

[70] KIVINEN, J., SMOLA, A. J., AND WILLIAMSON, R. C. Online learning with kernels. *IEEE transactions on signal processing 52*, 8 (2004), 2165–2176.

[71] KOKKINOS, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 6129–6138.

[72] KOLEN, J. F., AND POLLACK, J. B. Back propagation is sensitive to initial conditions. In *Advances in neural information processing systems* (1991), pp. 860–867.

[73] KRAUSE, J., STARK, M., DENG, J., AND FEI-FEI, L. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (2013), pp. 554–561.

[74] KRIZHEVSKY, A. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

[75] KRIZHEVSKY, A., AND HINTON, G. Learning multiple layers of features from tiny images. Tech. rep., Citeseer, 2009.

[76] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[77] KROGH, A., AND HERTZ, J. A. A simple weight decay can improve generalization. In *Advances in neural information processing systems* (1992), pp. 950–957.

[78] KRUSCHKE, J. K. Alcove: an exemplar-based connectionist model of category learning. *Psychological review 99*, 1 (1992), 22.

[79] KRUSCHKE, J. K. Human category learning: Implications for backpropagation models. *Connection Science 5*, 1 (1993), 3–36.

[80] KUMAR, A., AND DAUMÉ III, H. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning* (2012), Omnipress, pp. 1723–1730.

[81] LAVDA, F., RAMAPURAM, J., GREGOROVA, M., AND KALOUSIS, A. Continual classification learning using generative models, 2018.

[82] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[83] LEE, S.-W., KIM, J.-H., JUN, J., HA, J.-W., AND ZHANG, B.-T. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems* (2017), pp. 4652–4662.

[84] LENNIE, P. The cost of cortical computation. *Current biology 13*, 6 (2003), 493–497.

[85] LI, Z., AND HOIEM, D. Learning without forgetting. In *European Conference on Computer Vision* (2016), Springer, pp. 614–629.

[86] LI, Z.-W., ZHANG, J.-P., AND YANG, J. A heuristic algorithm to incremental support vector machine learning. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)* (2004), vol. 3, IEEE, pp. 1764–1767.

[87] LIN, L.-J. Reinforcement learning for robots using neural networks. Tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[88] LIU, B., WANG, M., FOROOSH, H., TAPPEN, M., AND PENSKY, M. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 806–814.

[89] LIU, J., AND JIA, Y. A lateral inhibitory spiking neural network for sparse representation in visual cortex. In *Advances in Brain Inspired Cognitive Systems* (Berlin, Heidelberg, 2012), H. Zhang, A. Hussain, D. Liu, and Z. Wang, Eds., Springer Berlin Heidelberg, pp. 259–267.

[90] LIU, X., MASANA, M., HERRANZ, L., VAN DE WEIJER, J., LOPEZ, A. M., AND BAGDANOV, A. D. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), IEEE, pp. 2262–2268.

[91] LOMONACO, V., DESAI, K., CULURCIELLO, E., AND MALTONI, D. Continual reinforcement learning in 3d non-stationary environments. *arXiv preprint arXiv:1905.10112* (2019).

[92] LOPEZ-PAZ, D., ET AL. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems* (2017), pp. 6470–6479.

[93] LOUIZOS, C., ULLRICH, K., AND WELLING, M. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems* (2017), pp. 3290–3300.

[94] MAJI, S., KANNALA, J., RAHTU, E., BLASCHKO, M., AND VEDALDI, A. Fine-grained visual classification of aircraft. Tech. rep., 2013.

[95] MALLYA, A., DAVIS, D., AND LAZEBNIK, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 67–82.

[96] MALLYA, A., AND LAZEBNIK, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 7765–7773.

[97] MANTE, V., SUSSILLO, D., SHENOY, K. V., AND NEWSOME, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature 503*, 7474 (2013), 78–84.

[98] MARC'AURELIO RANZATO, Y., AND LECUN, L. B. S. C. Y. A unified energy-based framework for unsupervised learning. In *Proc. Conference on AI and Statistics (AI-Stats)* (2007), vol. 24.

[99] MARTENS, J. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193* (2014).

[100] MCCLELLAND, J. L., MCNAUGHTON, B. L., AND O'REILLY, R. C. 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 (1995), 419.

[101] MCCLOSKEY, M., AND COHEN, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation 24* (1989), 109–165.

[102] MEUNIER, D., LAMBIOTTE, R., FORNITO, A., ERSCHE, K., AND BULLMORE, E. Hierarchical modularity in human brain functional networks. *Frontiers in Neuroinformatics 3* (2009), 37.

[103] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[104] MITCHELL, T. M. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey, 1980.

[105] MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[106] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S., AND HASSABIS, D. Human-level control through deep reinforcement learning. *Nature* (2014).

[107] MOORE, A. W., AND ATKESON, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning 13*, 1 (Oct 1993), 103–130.

[108] NETZER, Y., WANG, T., COATES, A., BISSACCO, A., WU, B., AND NG, A. Y. Reading digits in natural images with unsupervised feature learning.

[109] NGUYEN, A., YOSINSKI, J., AND CLUNE, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), IEEE, pp. 427–436.

[110] NGUYEN, C. V., LI, Y., BUI, T. D., AND TURNER, R. E. Variational continual learning. In *International Conference on Learning Representations* (2018).

[111] NILSBACK, M.-E., AND ZISSERMAN, A. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing* (Dec 2008).

[112] PENTINA, A., AND LAMPERT, C. H. A pac-bayesian bound for lifelong learning. In *ICML* (2014), pp. 991–999.

[113] PENTINA, A., AND LAMPERT, C. H. Lifelong learning with non-iid tasks. In *Advances in Neural Information Processing Systems* (2015), pp. 1540–1548.

[114] PERNICI, F., BARTOLI, F., BRUNI, M., AND BIMBO, A. D. Memory based online learning of deep representations from video streams. *CoRR abs/1711.07368* (2017).

[115] PERNICI, F., AND DEL BIMBO, A. Unsupervised incremental learning of deep descriptors from video streams. *ICMEW.2017.8026276.* (2017), 477–482.

[116] QUADRIANTO, N., PETTERSON, J., AND SMOLA, A. J. Distribution matching for transduction. In *Advances in Neural Information Processing Systems* (2009), pp. 1500–1508.

[117] QUATTONI, A., AND TORRALBA, A. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 413–420.

[118] RAMAPURAM, J., GREGOROVA, M., AND KALOUSIS, A. Lifelong generative modeling. *arXiv preprint arXiv:1705.09847* (2017).

[119] RAMASAMY, S., RAJARAMAN, K., KRISHNASWAMY, P., AND CHANDRASEKHAR, V. Online deep learning: growing rbm on the fly. *arXiv preprint arXiv:1803.02043* (2018).

[120] RANNEN, A., ALJUNDI, R., BLASCHKO, M. B., AND TUYTELAARS, T. Encoder based lifelong learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1320–1328.

[121] RATCLIFF, R. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological review 97*, 2 (1990), 285–308.

[122] REBUFFI, S.-A., KOLESNIKOV, A., SPERL, G., AND LAMPERT, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2017), pp. 2001–2010.

[123] RIBANDO, J. M. Measuring solid angles beyond dimension three. *Discrete & Computational Geometry 36*, 3 (2006), 479–487.

[124] ROBINS, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science 7*, 2 (1995), 123–146.

[125] RODRÍGUEZ, P., GONZALEZ, J., CUCURULL, G., GONFAUS, J. M., AND ROCA, X. Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967* (2016).

[126] ROLNICK, D., AHUJA, A., SCHWARZ, J., LILLICRAP, T. P., AND WAYNE, G. Experience replay for continual learning. *CoRR abs/1811.11682* (2018).

[127] ROLNICK, D., AHUJA, A., SCHWARZ, J., LILLICRAP, T. P., AND WAYNE, G. Unsupervised experience replay for continual learning. *arxiv:1811.11682*. (2018).

[128] ROSENFELD, A., AND TSOTSOS, J. K. Incremental learning through deep adaptation. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[129] ROYER, A., AND LAMPERT, C. H. Classifier adaptation at prediction time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1401–1409.

[130] RUECKL, J. Jumpnet: A multiple-memory connectionist architecture. *Proceedings of the 15 th Annual Conference of the Cognitive Science Society*, 24 (1993), 866–871.

[131] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., ET AL. Learning representations by back-propagating errors. *Cognitive modeling 5*, 3 (1988), 1.

[132] 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. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[133] RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A., BERNSTEIN, M., ET AL. Imagenet large scale visual recognition challenge. *International journal of computer vision 115*, 3 (2015), 211–252.

[134] RUSU, A. A., RABINOWITZ, N. C., DESJARDINS, G., SOYER, H., KIRKPATRICK, J., KAVUKCUOGLU, K., PASCANU, R., AND HADSELL, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[135] SAHOO, D., PHAM, Q., LU, J., AND HOI, S. C. Online deep learning: learning deep neural networks on the fly. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (2018), AAAI Press, pp. 2660–2666.

[136] SERRÀ, J., SURIS, D., MIRON, M., AND KARATZOGLOU, A. Overcoming catastrophic forgetting with hard attention to the task. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (2018), pp. 4555–4564.

[137] SHALEV-SHWARTZ, S., ET AL. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning 4*, 2 (2012), 107–194.

[138] SHAZEER, N., MIRHOSEINI, A., MAZIARZ, K., DAVIS, A., LE, Q., HINTON, G., AND DEAN, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.

[139] SHIN, H., LEE, J. K., KIM, J., AND KIM, J. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems* (2017), pp. 2990–2999.

[140] SHMELKOV, K., SCHMID, C., AND ALAHARI, K. Incremental learning of object detectors without catastrophic forgetting. In *The IEEE International Conference on Computer Vision (ICCV)* (2017).

[141] SILVER, D., HUBERT, T., SCHRITTWIESER, J., ANTONOGLOU, I., LAI, M., GUEZ, A., LANCTOT, M., SIFRE, L., KUMARAN, D., GRAEPEL, T., ET AL. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science 362*, 6419 (2018), 1140–1144.

[142] SILVER, D. L., AND MERCER, R. E. The task rehearsal method of life-long learning: Overcoming impoverished data. In *Conference of the Canadian Society for Computational Studies of Intelligence* (2002), Springer, pp. 90–101.

[143] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[144] SLOMAN, S. A., AND RUMELHART, D. E. Reducing interference in distributed memories through episodic gating. *Essays in honor of WK Estes 1* (1992), 227–248.

[145] SPRECHMANN, P., JAYAKUMAR, S., RAE, J., PRITZEL, A., BADIA, A. P., URIA, B., VINYALS, O., HASSABIS, D., PASCANU, R., AND BLUNDELL, C.

Memory-based parameter adaptation. In *International Conference on Learning Representations* (2018).

[146] SRIVASTAVA, N., HINTON, G., KRIZDHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[147] SRIVASTAVA, R. K., MASCI, J., KAZEROUNIAN, S., GOMEZ, F., AND SCHMIDHUBER, J. Compete to compute. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2310–2318.

[148] STREHL, A. L., AND LITTMAN, M. L. Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems* (2008), pp. 1417–1424.

[149] SUN, Y., WANG, X., AND TANG, X. Sparsifying neural network connections for face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4856–4864.

[150] TEREKHOV, A. V., MONTONE, G., AND O'REGAN, J. K. Knowledge transfer in deep block-modular neural networks. In *Conference on Biomimetic and Biohybrid Systems* (2015), Springer, pp. 268–279.

[151] THRUN, S., AND O'SULLIVAN, J. Clustering learning tasks and the selective cross-task transfer of knowledge. In *Learning to learn*. Springer, 1998, pp. 235–257.

[152] THRUN, S., AND PRATT, L. Learning to learn: Introduction and overview. In *Learning to learn*. Springer, 1998, pp. 3–17.

[153] TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological) 58*, 1 (1996), 267–288.

[154] VEDALDI, A., AND LENC, K. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia* (2015), ACM, pp. 689–692.

[155] VITTER, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS) 11*, 1 (1985), 37–57.

[156] WEISSTEIN, E. W. Lebesgue measure. from mathworld–a wolfram web resource.

[157] WELINDER, P., BRANSON, S., MITA, T., WAH, C., SCHROFF, F., BELONGIE, S., AND PERONA, P. Caltech-UCSD Birds 200. Tech. Rep. CNS-TR-2010-001, California Institute of Technology, 2010.

[158] XIONG, W., DU, B., ZHANG, L., HU, R., AND TAO, D. Regularizing deep convolutional neural networks with a structured decorrelation constraint. In *ICDM* (2016), pp. 519–528.

[159] XU, H., LIU, B., SHU, L., AND YU, P. S. Learning to accept new classes without training. *CoRR abs/1809.06004* (2018).

[160] XU, J., AND ZHU, Z. Reinforced continual learning. In *NeurIPS* (2018), pp. 907–916.

[161] XUE, J., LI, J., AND GONG, Y. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech* (2013), pp. 2365–2369.

[162] XUE, Y., LIAO, X., CARIN, L., AND KRISHNAPURAM, B. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research 8*, Jan (2007), 35–63.

[163] YAO, L., AND MILLER, J. Tiny imagenet classification with convolutional neural networks. *CS 231N* (2015).

[164] YOON, J., YANG, E., LEE, J., AND HWANG, S. J. Lifelong learning with dynamically expandable networks.

[165] YU, Y., MIGLIORE, M., HINES, M. L., AND SHEPHERD, G. M. Sparse coding and lateral inhibition arising from balanced and unbalanced dendrodendritic excitation and inhibition. *Journal of Neuroscience 34*, 41 (2014), 13701–13713.

[166] ZEILER, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701* (2012).

[167] ZEILER, M. D., RANZATO, M., MONGA, R., MAO, M., YANG, K., LE, Q. V., NGUYEN, P., SENIOR, A., VANHOUCKE, V., DEAN, J., ET AL. On rectified linear units for speech processing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), IEEE, pp. 3517–3521.

[168] ZENKE, F., POOLE, B., AND GANGULI, S. Improved multitask learning through synaptic intelligence. In *Proceedings of the International Conference on Machine Learning (ICML)* (2017).

[169] ZHANG, J., ZHANG, J., GHOSH, S., LI, D., TASCI, S., HECK, L. P., ZHANG, H., AND KUO, C. J. Class-incremental learning via deep model consolidation. *CoRR abs/1903.07864* (2019).

[170] ZHANG, Y., AND YANG, Q. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* (2017).

Curriculum

Rahaf Aljundi was born on 1988 in Syria. After finishing her secondary school, she studied computer science engineering at Damascus university, Syria and specialized in artificial intelligence (AI). She graduated among the top ten students and received a prize for her graduation project from ICT incubator Syria. After graduation, she worked in industry as a software engineer while being a part time teaching assistant and master student in the AI master program, both at Damascus University. In 2012, She was granted a PhD scholarship from her home university. However, in 2013, she moved to Europe to start a master program in Machine Learning and Data Mining which was a joint program between Jean Monnet University, France and University of Alicante, Spain. Her research during the master's degree was focused on Domain Adaptation for both computer vision and medical imaging. She was ranked first with honours for the work of her master thesis. In October 2015, she joined the Visics group, KU Leuven as a PhD student under the supervision of prof. Tinne Tuytelaars. In 2016 she received an FWO scholarship for her PhD research which was renewed on 2018. From Nov 2018 to Feb 2019 she was on a research visit to Montreal institute for learning Algorithms (Mila) working on Continual Learning with dr. Min Lin and prof. Yoshua Bengio. During her PhD, she worked on continual learning and published papers in top conferences in Machine Learning and Computer vision including Neurips, ICLR, CVPR, ICCV, and ECCV. She also served as a reviewer for top machine learning and computer vision conferences and journals.

List of Publications

Book chapters

- Unsupervised Domain Adaptation based on Subspace Alignment. B Fernando, R Aljundi, R Emonet, A Habrard, M Sebban and T Tuytelaars, Domain Adaptation in Computer Vision Applications; Springer, Cham, 2017. 81-94

International peer reviewed conferences

- Gradient based Sample Selection for Online Continual Learning. R Aljundi, M Lin, B Goujaud, and Y Bengio in 2019 Conference on Neural Information Processing Systems (Neurips'19).
- Online Continual Learning with Maximally Interfered Retrieval. R Aljundi*, L Caccia*, E Belilovsky*, M Caccia*, M Lin, L Charlin, T Tuytelaars in 2019 Conference on Neural Information Processing Systems (Neurips'19).
- Task-free continual learning. R Aljundi*, K Kelchtermans*, T Tuytelaars in Computer Vision and Pattern Recognition (CVPR'19).
- Selfless Sequential Learning. R Aljundi, M Rohrbach and T Tuytelaars in International Conference on Learning Representations (ICLR'19).
- Exploring the Challenges towards Lifelong Fact Learning. M Elhoseiny, F Babiloni, R Aljundi, M Rohrbach and T Tuytelaars in Asian Conference on Computer Vision (ACCV'18).
- Memory Aware Synapses: Learning what (not) to forget. R Aljundi, F Babiloni, M Elhoseiny, M Rohrbach and T Tuytelaars in European Conference on Computer Vision (ECCV'18).
- Encoder Based Lifelong Learning AR Triki*, R Aljundi*, M Blaschko and T Tuytelaars in International Conference on Computer Vision (ICCV'17).

- Expert Gate: Lifelong Learning with a Network of Experts. R Aljundi, P Chakravarty, T Tuytelaars in Computer Vision and Pattern Recognition (CVPR'17).
- Who's that Actor? Automatic Labelling of Actors in TV series starting from IMDB Images. R Aljundi*, P Chakravarty*, T Tuytelaars in Asian Conference on Computer Vision (ACCV'16).
- Landmarks-based Kernelized Subspace Alignment for Unsupervised Domain Adaptation R Aljundi, R Emonet, D Muselet and M Sebban In Computer Vision and Pattern Recognition (CVPR'15).

International peer reviewed workshops

- Lightweight Unsupervised Domain Adaptation by Convolutional Filter Reconstruction. R Aljundi, T Tuytelaars in TASK-CV: Transferring and Adapting Source Knowledge in Computer Vision workshop (ECCV'16).
- Transfer Learning for Prostate Cancer Mapping Based on Multicentric MR imaging databases. R Aljundi, J Lehaire, F Prost-Boucle, O Rouvière and C Lartizien In Machine Learning meets Medical Imaging workshop (ICML'15).

Under review

- Continual learning: A Comparative Study on How to Defy Forgetting in Classification Tasks. M De Lange, R Aljundi, M Masana, S Parisot, X Jia, A Leonardis, G Slabaugh, T Tuytelaars. Submitted as a journal paper to TPAMI.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
ESAT - PSI
Kasteelpark Arenberg 10 - box 2441
B-3001 Leuven
rahaf.aljundi@esat.kuleuven.be
http://homes.esat.kuleuven.be/ raljundi/