Continual Learning for Efficient Machine Learning

Arslan Chaudhry
New College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2020

To my parents (Ammi and Abbu) for everything.

Declaration

I hereby declare that this thesis (and the work presented in it) is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy, and that this thesis is entirely my own work based on publications in collaboration with the co-authors where due acknowledgement is made.

November 2020

Arslan Chaudhry

Acknowledgements

As I sit down to write this section, which I purposefully left at the very end, I realize that the number of people I need to acknowledge far exceeds the limits of a reasonably sized acknowledgment section. After all, the culmination of this thesis is not just due to the support of the people and institutions I came across in the last four years, but every individual, who taught me at any time, deserves my gratitude. I thank the beautiful humans, be in school, college, or university, who touched my life before coming to Oxford.

It goes without saying that this thesis would not have been possible without the continuous support of my supervisor, Philip Torr. However, perhaps more than that, I like to thank Phil for giving me an opportunity to pursue a DPhil in his group and teaching me the basics of research when most labs in the world expect fluency in research a priori. I am also indebted to Puneet K. Dokania who acted as my secondary supervisor and patiently taught me how to be a better researcher.

I am extremely lucky to have collaboration opportunities outside Oxford during my DPhil. I am thankful to Marc'Aurelio Ranzato, Marcus Rohrbach, Mohamed Elhoseiny, Albert Gordo, and David Lopez-Paz for hosting me multiple times at the Facebook AI lab. I had the most productive research experience during these visits. I am also thankful to Srikumar Ramalingam, Andreas Veit, Sadeep Jayasumana, and Aditya Krishna Menon for hosting me at Google research.

I would like to express my deepest gratitude to the present and former members of the Torr Vision Group for keeping the first floor of the Information Engineering Building abuzz with the excitement of new ideas. I am particularly thankful to Thalaiyasingam Ajanthan. His singular work ethic and scientific rigor have inspired me the most during the last four years. Naeemullah Khan has been a constant source of encouragement in the last two years in and outside the lab. His impeccable analytical skills along with the unique sense of humor made the last two winters in Oxford a bit more bearable. I also thank Nantas Nardelli for

uplifting me, often without substance, and encouraging me to apply for research internships and jobs. His energy and positivity are infectious. The occasional lunches, coffee chats, and sometimes long runs were the much needed punctuation in a typical DPhil day. I thank Sebastien, Qizhu, Namhoon, Viveka, Arnab, and Csabi for providing these breaks.

Life in Oxford would not have been possible without the support of Mohsin, Sarah, Amal, Adeel, Talha, and Vikaran. The last four years were the most transformative of my life and these six, more than anyone, deserve that credit for it. I thank all of you for everything. I can't ask for a better friend than Talha Ghaffar who patiently listened to all my personal and professional worries from across the Atlantic and, importantly, never asked me what to do.

I am thankful to my partner, Summiya, for being a constant source of love and inspiration for the last one and a half years. A lot in my life is better because of her perspective. I owe my deepest gratitude to my parents (Ammi and Abbu) and Adnan. It would not have been possible to survive the past four years without the context they provide.

Finally, I am grateful to the Rhodes Trust for funding the first three years of my DPhil and Amazon for the last one year.

Abstract

Deep learning has enjoyed tremendous success over the last decade, but the training of practically useful deep models remains highly inefficient both in terms of the number of weight updates and training samples. To address one aspect of these issues, this thesis studies the continual learning setting whereby a model utilizes a sequence of tasks leveraging previous knowledge to learn new tasks quickly. The main challenge in continual learning is to keep the model from catastrophically forgetting the previous information when updating it for a new task.

Towards this, firstly this thesis proposes a continual learning algorithm that preserves previous knowledge by regularizing the KL-divergence between the conditional likelihoods of two successive tasks. It is shown that this regularization imposes a quadratic penalty on the network weights based on the curvature at the minimum of the previous task.

Second, this thesis presents a more efficient continual learning algorithm, utilizing an episodic memory of past tasks as a constraint such that the loss of episodic memory does not increase when a weight update is made for a new task. It is shown that using episodic memory to constrain the objective is more effective than regularizing the network parameters. Furthermore, to increase the speed of learning of new tasks, the use of compositional task descriptors using a joint embedding model is proposed that greatly improves the forward transfer.

The episodic memory-based continual learning objective is then simplified by directly using memory in the loss function. Despite its tendency to memorize the data present in the tiny episodic memory, the resulting algorithm is shown to generalize better than the one where memory is used as a constraint. An analysis is proposed that attributes this surprising generalization to the regularization effect brought by the data of new tasks.

This algorithm is then used to learn continually from synthetic and real data. For this, a method is proposed that generates synthetic data points

for each task by optimizing the forgetting loss in hindsight on the replay buffer. A nested optimization objective for continual learning is devised that effectively utilizes these synthetic points to reduce forgetting in memory-based continual learning methods.

Finally, this thesis presents a continual learning algorithm that learns different tasks in nonoverlapping feature subspaces. It is shown that minimizing the overlap by keeping the subspaces of different tasks orthogonal to each other reduces the interference between the representations of these tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is written with the integrated format in mind, whereby published work in peer-reviewed conferences and workshops is reformatted as different chapters of this manuscript. This chapter provides an overview of the integrated thesis. Section 1.1 briefly summarizes the success of deep learning, a subfield of machine learning, and some of the challenges around its efficiency – setting the context for the remainder of the thesis. Section 1.2 then introduces continual learning as a possible choice for efficient machine learning along with its own implementation difficulties in deep learning models. Sections 1.3 and 1.4 provide an outline and main contributions of the thesis, respectively. Finally, section 1.5 lists the publications that constitute the main chapters of this thesis.

1.1 Deep Learning: the good and bad

The quest for formal systems capable of reasoning dates to ancient philosophers and mathematicians. In the mid of the twentieth century, the term *Artificial Intelligence* (AI) was coined by McCarthy et al. [91] to enable the 'intelligent behaviour' specifically in computing machines (a.k.a computers). This led to myriad inquiries to develop intelligent machines using different techniques; *Symbolic AI* [51, 24]

pertains to enabling intelligent behaviour by manipulating symbols. *Evolutionary algorithms* takes inspiration from biological evolutionary processes to develop intelligent machines [13]. *Machine Learning*, arguably the most promising field of AI, utilizes statistical modelling to develop algorithms that improve automatically by observing data.

More recently, deep learning [79, 47], a subfield of machine learning that uses multiple layer neural networks to automatically extract features from data, has provided a series of breakthroughs in multiple domains owing, in part, to the availability of large datasets and faster compute. In computer vision, deep learning models have been shown to provide superior performance than other approaches in a variety of recognition tasks including but not limited to, object classification, [69, 53] (figure 1.1), object detection [120, 138], object tracking [22, 148], semantic segmentation [33, 11, 54] *etc*. In natural language processing, deep learning models are the reigning state-of-the-art in a variety of applications including text classification [154, 159, 72, 96], translation [14, 88, 144, 76], and dialogue systems [130, 113], to name a few. In game play, deep learning models have been shown to achieve superhuman performance in different scenarios. For example, in perfect information games, such as Go, Chess or Shogi, Silver et al. [131] showed that deep neural networks with the help of self-play and tree search methods can become the strongest players in each game. Similarly, in imperfect information games, such as poker, Brown and Sandholm [26] showed that their algorithm, Pluribus, was able to consistently beat human players in six-player no-limit Texas hold'em poker. Likewise, in the fields of robotics [125, 124], drug discovery [135], and medical imaging [103] deep learning has enjoyed tremendous success.

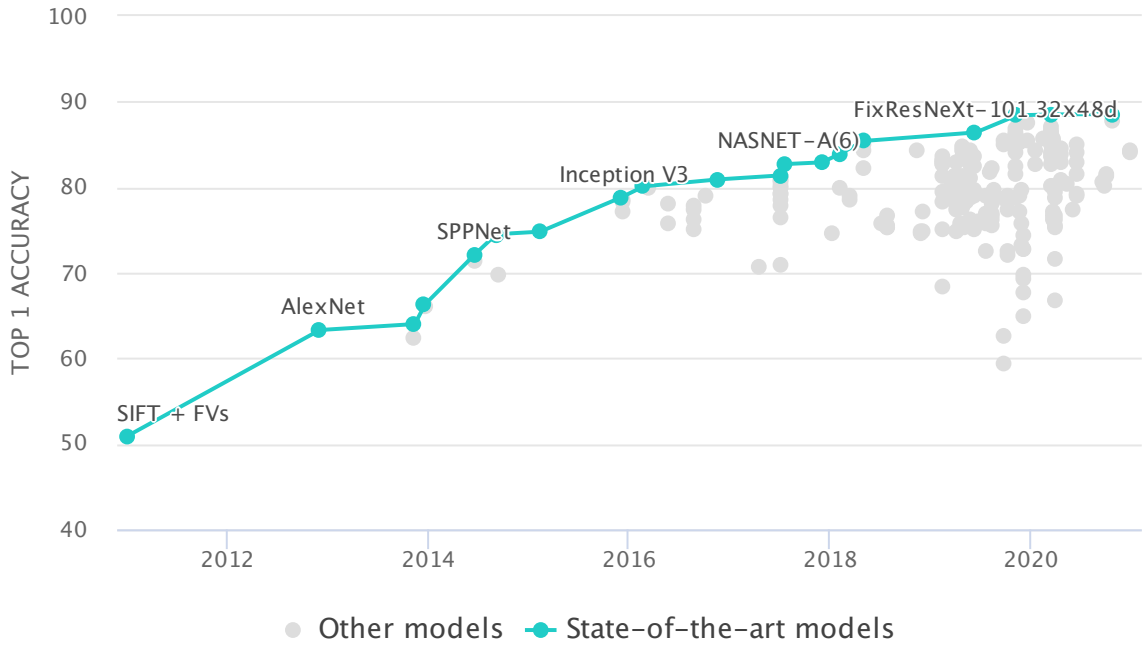


Figure 1.1: Performance improvement in ImageNet [117] over the last decade. Figure from https://paperswithcode.com/.

Learning efficiency challenges

The recipe of deep learning that has worked well and brought breakthroughs in different domains is *supervised learning* that can be summarized in the following three steps. The first step is to collect a large training dataset of input and output pairs. Second, define an application-specific objective *e.g.*, an empirical loss function, and finally, the third step is to repeatedly train – sampling from the dataset and optimizing the objective – over many iterations. While this recipe has allowed deep neural networks to prove to be very powerful function approximators, the success is not without cost[1]. The associated cost, among other things, is the requirement of copious amounts of supervision required for training deep models, large training times (slow convergence), and the need to update the model from scratch when the training environment is changed. All these ailments make the deep learning models *inefficient* and preclude their applicability in diverse real-world scenarios.

[1]In the remainder, we use machine learning, deep learning, and supervised learning interchangeably.

The inefficiency of deep learning models can be viewed at different axes:

Sample inefficiency Current deep learning models require thousands of examples per concept, *e.g.*, ImagetNet [36] has roughly 1000 examples per class and CIFAR-10 [68] has 5000 examples per class (see figure 1.2 for other datasets), to learn a concept. The collection of these large datasets [38, 36, 73] is an expensive and laborious task. Furthermore, annotating these datasets for supervised training is a cumbersome task. For example, it takes roughly one second per class or object category to annotate an image for object classification [101]. Even worse, Bearman et al. [17] pointed out that for semantic segmentation tasks it takes almost four minutes on average to annotate a single image. This shows that collecting and labelling a dataset for a plethora of concepts that exist in the real world is an impossible task making the standard supervised learning difficult to scale. Therefore, we need learning algorithms that are more sample efficient.

Slow convergence Deep learning, as implemented today, requires multiple revisits of the same dataset – multiple epochs of training – to find solutions that perform well. This not only puts the constraint on learning algorithms to store these large datasets offline, which could be prohibitively expensive memorywise, but also results in longer training times. For example, He et al. [53] reported that they trained a deep learning model (ResNet-50) on 8 Tesla P100 GPUs for 29 hours to achieve the desirable performance. This slow convergence of deep learning towards the optimal solution makes the machine learning models unfit to work in real-time. Therefore, we should strive for machine learning models that can converge to good solutions in a few training steps, preferably by visiting the dataset only once.

Slow Adaption If new data arrives, the standard practice in machine learning is to combine the new data with the previous one and retrain the model from scratch

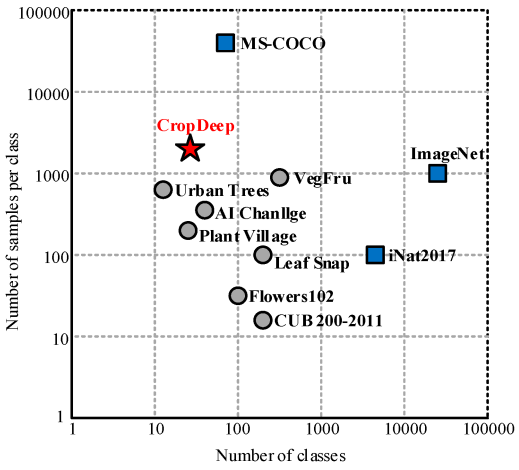


Figure 1.2: Comparison of different computer vision datasets in terms of number of samples per class and total number of classes. Figure from Zheng et al. [158].

or finetune the previous model. In both cases, even the concepts that had already been learned from previous data need to be relearned. This makes the current deep learning models highly static in application and any attempt to use them in dynamic data environments prohibitively expensive to train. We need to search for learning algorithms that can train efficiently in such dynamically changing environments.

1.2 Continual learning

We propose to study continual learning, as a possible solution for the deep learning problems around efficiency as described in the previous section. Continual learning, also referred to as lifelong learning, incremental learning, or sequential learning, studies the problem of learning from a stream of data arising from changing input and output distributions associated with different tasks [112, 140]. The idea is to divide the original large learning problem into smaller tasks and then learn from these tasks in a continual manner, one after the other. This could potentially solve all three main efficiency issues of deep learning as described in the previous section. First, while a typical deep learning model needs to collect and store a large dataset for the original learning problem, dividing the problem into

smaller tasks and learning from them in a sequence obviates the full dataset storage requirement. Second, as the model trains continually, it should be able to *quickly* learn new tasks by leveraging past experiences. This not only reduces the dataset size – by reducing the required number of samples per concept – but also improves the convergence speed for future tasks. Third, as the model is trained continually from the beginning, it is well suited to adapt to any dynamically changing environments.

Challenges in continual learning

The main challenge in continual learning is that of *catastrophic forgetting* [93, 107]: when a new task is learned, it causes the sudden and complete disappearance of the previous information. By wiping out the previously learned knowledge, catastrophic forgetting precludes the machine learning model from using this knowledge to quickly adapt to future tasks. We note here that the goal is not to make a machine learning model to remember every detail it had seen in the past. After all, if the model has not seen a data point for a while, the data could be unimportant and the machine learning model can forget it gracefully. Even healthy adult humans learn and unlearn gradually over time [15]. However, humans do not demonstrate catastrophic forgetting of previous knowledge and our aim is to achieve similar in machine learning models.

The ability of humans to learn sequentially is attributed to the separation of the hippocampus and neocortex in the human brain (mammalian brain). McClelland et al. [92] argued that, in humans, new information is first learned in the hippocampus and then gradually consolidated in the neocortex for long-term storage. Through these two complementary learning systems, the human brain avoids catastrophic forgetting. However, the mere presence of these two systems does not guarantee to prevent catastrophic forgetting. For example, French and Ferrara [46]

demonstrated that rats undergo catastrophic forgetting in time-duration tasks despite having both a hippocampus and a neocortex. They attributed this forgetting in rats to noninvolvement of the hippocampal/neocortical loop in time-duration tasks.

The inquires into catastrophic forgetting in neural networks dates to the late 1980's and early 1990's (for a detailed review, refer to [45]). It was thought, and rightly so, that the catastrophic forgetting in neural networks is due to overlapping hidden representations [44]. Several algorithms were developed in the early 1990s to reduce this overlap. Some of the algorithms [44, 70, 97] relied on explicitly modifying the hidden-layer representations to reduce the overlap, while others [83] orthogonalize the input representations to reduce this interference. More recently, Goodfellow et al. [48], Kirkpatrick et al. [67], Li and Hoiem [85] revived the study of catastrophic forgetting in deep neural networks.

The issue of not forgetting the previous tasks and learning a new task is connected with a more general framework of stability-plasticity dilemma [49] in neural networks. The stability refers to preserving the previous knowledge, and plasticity refers to updating the model to learn new tasks. Increasing the plasticity of the model leads to forgetting (less stability). Similarly, trivially increasing the stability can prevent the model from learning new tasks. A continual learning algorithm finds the right balance between these two competing objectives depending on the application. The development of such efficient continual learning models is the focus of this thesis.

1.3 Thesis outline

The other chapters of this thesis are outlined below:

Chapter 2. This chapter reviews the continual learning desiderata, training and evaluation setup, benchmarks, and evaluation metrics that are used throughout the thesis. It also compares continual learning with related machine learning fields.

Chapter 3. This chapter presents a continual learning method that preserves previous knowledge by regularizing the network parameters. The practical issues of different training and evaluation setups are also discussed in this chapter.

Chapter 4. This chapter introduces a more efficient continual learning method that uses episodic memory as an optimization constraint. A joint embedding model utilizing task attributes is presented that effectively transfers knowledge to future tasks. The efficacy of regularisation-based algorithms, studied in the previous chapter, is also analyzed in a single-epoch setup.

Chapter 5. In the previous chapter, an episodic memory-based continual learning algorithm is presented that uses memory as constraints. In this chapter, the optimization objective is simplified and an algorithm is presented that directly trains on the episodic memory. The proposed algorithm generalizes better than the one presented in chapter 4. The memorization effect of the algorithm is analyzed in this chapter.

Chapter 6. In chapter 5, we see that direct training on the episodic memory gives the best generalization. In this chapter, an algorithm is presented that can synthesize the samples to write into memory. Additionally, a nested optimization objective is proposed for continual learning.

Chapter 7. In this chapter, an algorithm is developed that learns different tasks in different orthogonal subspaces. It is shown that by reducing the overlap between different subspaces, catastrophic forgetting can be effectively reduced.

Chapter 8. Finally, we conclude in chapter 8 by summarizing the contributions presented in this thesis. A brief discussion about future directions is also presented.

Note that this is an integrated thesis, and that each of chapters 3–7 has been published at leading peer-reviewed conferences in machine learning and computer vision. These papers have only been reformatted, and the supplementary material has been included as an appendix at the end of each chapter. Each chapter is self-contained with its own related work section centred around the contribution of the paper. These contributions are now described next.

1.4 Contributions

The main contribution of this thesis is the development of efficient continual learning algorithms with practical constraints. We briefly summarize the contributions as follows:

Chapter 3: Parameter Regularization-based Continual Learning: Riemannian Walk. In this chapter, we propose a generalization of two continual learning algorithms based on parameter regularization, EWC [67] and PI [156], derived from a KL-divergence perspective. The proposed algorithm is dubbed as Riemannian Walk (RWALK) and it provides a better trade-off between different continual learning metrics than EWC and PI. We further propose two additional evaluation metrics for continual learning, maximum average forgetting, and intransigence, where the former measures the performance drop of a task from its peak accuracy and the latter measures the performance gap of a task from when the same task is learned without any constrain on learning *i.e.* maximizing plasticity. These two metrics effectively quantify the stability-plasticity dilemma described in section 1.2. The chapter also provides the differing learning complexity of various evaluation

measures *i.e.* single-head and multi-head evaluations, that were missing from the literature prior to the work's publication.

Chapter 4: Efficient Continual Learning: Averaged Gradient Episodic Memory. This chapter builds on the work of Lopez-Paz and Ranzato [87] and proposes a continual learning algorithm, Averaged Gradient Episodic Memory (A-GEM), where the episodic memory of previous tasks is used as an optimization constraint that prevents the loss of previous tasks from increasing when the model is trained on a new task. In particular, we use the average gradient of episodic memory as an optimization constraint and show that despite being more efficient, both in time and memory, our approach performs as well as when all previous tasks are used as constraints. We also propose a joint embedding model that uses task descriptions (class attributes) to improve forward transfer in continual learning algorithms. A metric to measure the forward transfer, Learning Curve Area (LCA), is also presented in this chapter.

Chapter 5: Continual Learning by Directly Replaying the Episodic Memory. This chapter simplifies the optimization objective developed in the previous chapter by adding the loss on episodic memory directly into the training objective instead of using it as an optimization constraint. To adhere to the practicality of continual learning algorithms, the size of the episodic memory is kept small. We show that, for the same size of episodic memory, directly replaying memory, dubbed as experience replay, generalizes better than when the memory is used as a constraint. This finding is rather surprising and goes against the intuition that over-parameterized models tend to overfit to small amounts of training data. We analyze how memorization in continual learning can be beneficial and that similar tasks can act as implicit data-dependent regularizers preventing overfitting to the small memory.

Chapter 6: Synthesizing and Anchoring Memory Samples for Continual Learning. This chapter builds on the previous chapter where we show that direct training on episodic memory yields the strongest generalization. The focus of this chapter is to synthesize the samples to store in the episodic memory. We propose an algorithm, Hindsight Anchor Learning (HAL), that generates synthetic samples, called anchors, that prevent the forgetting of previous tasks more effectively than real data samples. We further provide a nested optimization objective for continual learning that uses these anchors to regularize the training on a new task. We analytically show that this anchoring objective improves the forwards transfer in continual learning by maximizing the inner product between the gradients of different tasks.

Chapter 7: Continual Learning in Low-rank Orthogonal Subspaces. This chapter provides a continual learning algorithm that learns different tasks in nonoverlapping vector subspaces. We propose an algorithm that partitions the feature vector space into orthogonal subspaces and projects the features of each task into a task-specific subspace. The gradients in the earlier network layers are kept from catastrophically interfering with each other by learning isometric transformations of the gradients coming from these nonoverlapping subspaces. The isometric transformations are learned by posing the network training as an optimization problem over the Stiefel manifolds. The proposed algorithm reduces forgetting significantly with and without episodic memory in different continual learning benchmarks.

1.5 Publications

The contributions described in the previous section are based on the following publications:

Chapter 3

- **Arslan Chaudhry**, Puneet K. Dokania, Thalaiyasingam Ajanthan, Philip H. S. Torr. "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence". *European Conference on Computer Vision (ECCV)*, 2018.
- **Code**: https://github.com/facebookresearch/agem

Chapter 4

- **Arslan Chaudhry**, Marc'Aurelio Ranzato, Marcus Rohrbach, Mohamed Elhoseiny. "Efficient Lifelong Learning with A-GEM". *International Conference on Learning Representations (ICLR)*, 2019.
- **Code**: https://github.com/facebookresearch/agem

Chapter 5

- **Arslan Chaudhry**, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, Marc'Aurelio Ranzato. "On Tiny Episodic Memories in Continual Learning". *International Conference on Machine Learning (ICML) Workshop on Multitask and Reinforcement Learning*, 2019.
- **Code**: https://github.com/facebookresearch/agem

Chapter 6

- **Arslan Chaudhry**, Albert Gordo, Puneet K. Dokania, Philip H. S. Torr, David Lopez-Paz. "Using Hindsight to Anchor Past Knowledge in Continual Learning". *Association for the Advancement of Artificial Intelligence (AAAI)*, 2021.

Chapter 7

- **Arslan Chaudhry**, Naeemullah Khan, Puneet K. Dokania, Philip H. S. Torr. "Continual Learning in Low-rank Orthogonal Subspaces". *Neural Information Processing Systems (NeurIPS)*, 2020.

- **Code**: https://github.com/arslan-chaudhry/orthog_subspace

The following publication on the related topic is also made during this thesis:

- **Arslan Chaudhry**, Puneet K. Dokania, Philip H. S. Torr. "Discovering Class-Specific Pixels for Weakly-Supervised Semantic Segmentation". *British Machine Vision Conference (BMVC)*, 2017.

- **Code**: https://github.com/arslan-chaudhry/dcsp_segmentation

Chapter 2

Background

This chapter provides a brief review of continual learning. Section 2.1 formalizes the problem of continual learning and section 2.2 provides the desiderata for such a learning setting. Section 2.3 compares continual learning with related machine learning fields. Section 2.4 describes the continual learning setup, benchmarks and evaluation protocols considered in this thesis. Note that the main contributions of this thesis are provided in chapters 3–7 where each chapter contains a literature review relevant to its contribution.

2.1 Continual learning: a sequential approach to machine learning

In supervised machine learning, the starting point is to assume the availability of a labeled dataset in the form of input-output pairs, $\mathcal{D} = \{x_i, y_i\}_{i=1}^{N}$, sampled identically and independently (iid) from a fixed joint probability distribution $P(X, Y)$. The objective is to learn a model $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, parameterized[1] by θ that predicts a target vector ($y \in \mathcal{Y}$) for any input ($x \in \mathcal{X}$). This is often achieved by the principle of

[1]We assume parametric models here.

Empirical Risk Minimization (ERM) [143] whereby the population risk on $P(X, Y)$ is approximated by an empirical risk of the form:

$$\mathbb{E}[\ell(f(x), y)] = \int \ell(f(x), y) dP(X, Y) \approx \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), y_i),$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is an application specific loss function, also called objective, risk or energy function. Under some assumptions (realizability and PAC), the generalization error (the error between the population and empirical risk) of the ERM can be bounded and it asymptotically ($N \to \infty$) converges to zero, refer to Vapnik [143] for details. The expected loss is subsequently minimized by an optimizer:

$$\arg\min_{\theta} \mathbb{E}[\ell(f(x), y)].$$

Neural networks or multiple layer perceptrons (MLP) [150], that were around for decades, have consistently been shown in the last decade to be an appropriate choice for the family of functions $f(\cdot)$ parameterized by θ. Figure 2.1 shows an example of a two-layer neural network that takes an input $\mathbf{x} \in \mathbb{R}^n$, generates a hidden representation $\mathbf{h} = \sigma(W_1 x + b_1)$ and then produces an output $\mathbf{y} = W_2 \mathbf{h} + b_2$, where $\sigma(\cdot)$ is a non-linearity, $\mathbf{h} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^C$, and $\{W_1, W_2\}$, $\{b_1, b_2\}$ are weight matrices and bias vectors of appropriate dimensions of the network, respectively. The presence of a nonlinearity $\sigma(\cdot)$ in the hidden layers makes the overall training objective of the neural network nonconvex, and iterative gradient descent approaches are relied upon to minimize the objective. The backpropagation algorithm [116] is most frequently used to iteratively update the neural network weights to optimize the learning objective. Krizhevsky et al. [69], He et al. [53], Vaswani et al. [144], Silver et al. [131] showed that very deep variants of neural networks, similar to the one shown in figure 2.1, can achieve remarkably high performance in different domains.

While supervised learning with neural networks has been quite successful in the

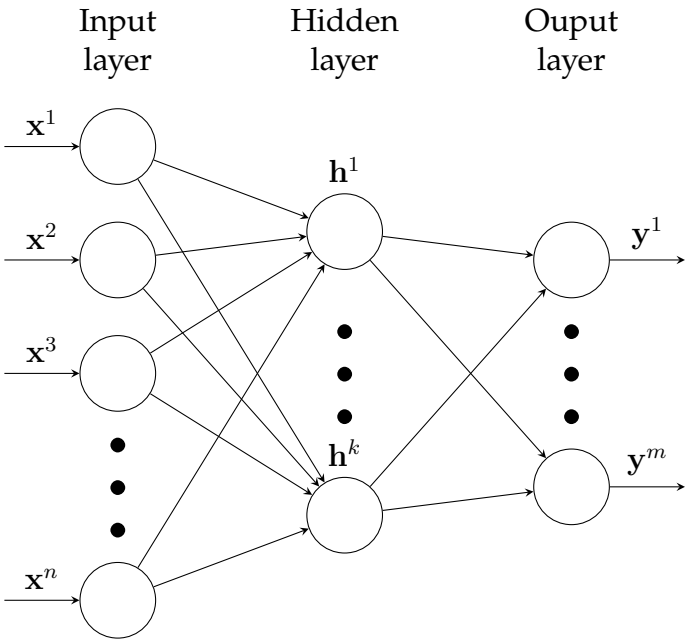


Figure 2.1: A two-layer neural network

last decade, it suffers from a myriad of efficiency issues as discussed in section 1.1. Continual learning, also called lifelong [140], sequential or incremental learning, gets around those efficiency issues by learning from multiple tasks. More formally, in a continual learning setup, the learning problem is divided into a sequence of tasks $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \cdots\}$, the length of which may or may not be known a priori. Each task k consists of N_k data triplets drawn from a task probability distribution, $(x^k, y^k, t^k) \sim P_k$, where x, y and $t \in \mathcal{T}$, are the input, output and task description, respectively.[1] The goal is to learn a function $f : \mathcal{X} \times \mathcal{T}_t \rightarrow \mathcal{Y}$ by minimizing the multitask error:

$$\frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\ell(f(x_i^t, t_i), y_i^t). \tag{2.1}$$

The constraint posed to the learning is that all tasks are *not* observed at once, and, instead, received in a sequence *i.e.* when accessing the dataset of task t, the datasets of other tasks ($k \neq t$) are not available, making the overall problem non-iid. This breaks the implementation of the empirical risk defined in equation 2.1

[1] We consider a continual learning setup where a task is always known.

that assumes all data samples are drawn iid from a fixed probability distribution. A trivial approach to solve this problem would be to store the datasets of all previous tasks and minimize equation 2.1 from scratch whenever a new task arrives. This approach would suffer from the same efficiency issues as that of the original supervised learning (see the discussion in section 1.1). Therefore, for this learning setting to be useful, the model needs to be constrained in terms of the amount of compute and memory required.

2.2 Desiderata of continual learning

An ideal continual learner needs to follow certain properties in order for the continual learning setting to be useful for efficient machine learning. Below we describe these desiderata:

Backward transfer/ less forgetting. The continual learner should not forget the previous tasks, at least not catastrophically, as it experiences new tasks. The tendency of the model to forget previous tasks is described as negative backward transfer. On the contrary, a continual learning algorithm should exhibit a positive backward transfer *i.e.*, it should improve the performance of previous tasks that are similar to the new task.

Forward transfer/ fast adaptation. As the model sees new tasks, it should be able to leverage knowledge from previous tasks to learn new tasks quickly *i.e.*, it should transfer knowledge in the forward direction. The backward and forward transfers are inherently linked; a continual learner that is less forgetful (less negative backward transfer) should be able to transfer more knowledge to future tasks.

Small memory. The memory footprint of a continual learning algorithm should not grow super-linearly with the number of tasks. Note, this constrains both the model size and the memory required to store the dataset of previous tasks. While a continual learner can store a small constant memory, for keeping the important information from the past, it should not store the full datasets of previous tasks.

Real time. The continual learning algorithm learns from a stream of data and the learner itself does not have a control on the speed at which the data arrives. Therefore, a continual learning algorithm should not involve in computations that are not real-time or where the learner can miss the incoming data while still training on the previous data.

Single epoch training/ fast convergence. The continual learner should learn from the data by seeing an example only once. It should not rely on offline multiple-epochs training.

Exploit the information in the sequence. If the data is not randomly shuffled, then the continual learning algorithm should exploit the information present in the sequence to improve convergence.

2.3 Comparison to related machine learning fields

The ideas of learning from multiple tasks, transferring knowledge from one task or domain to the next, faster convergence to new tasks, or learning from a stream of data are also studied in other machine learning fields. Below we describe these fields and compare them with continual learning.

Transfer learning. In the most general setting, transfer learning refers to quickly learning – learning with less training data or less training steps – a new task on a

new domain by using the information from a model trained on another task in a different domain. Similar to continual learning, forward transfer from the previous domain to the new is sought in transfer learning. Unlike continual learning, the performance of a previous domain or task does not need to be retained, nor there is a continual adaptation to future tasks or domains. Finetuning is an example of transfer learning where a model trained on a large dataset is adapted to a much smaller dataset by only changing the last layer weights.

Domain adaptation. Domain adaptation is a subclass of transfer learning in which the underlying task in both domains is the same. The model is adapted from a labeled source domain to an unlabeled target domain [119, 142]. An example of domain adaptation would be to adapt an object detector, trained on labelled images collected during daylight to night time.

Multitask learning. Multitask learning [27] pertains to learning from multiple tasks, potentially coming from different distributions, by minimizing the objective defined in equation 2.1. Multitask learning allows for the discovery of a shared structure across all tasks, making the model to achieve greater performance and efficiency than what was possible when the tasks are trained individually. Contrary to continual learning, multitask learning assumes simultaneous access to all the tasks making offline training and zero forgetting a possibility. Note that it is not always possible to find a good solution for all tasks in multitask learning [55, 155].

Meta learning. Meta learning or learning to learn is a learning paradigm that allows the learning procedure to improve itself with experience [122, 19, 8, 108]. Recently, meta-learning is used in the context of learning generic knowledge from numerous tasks, consisting of a small set of training examples, and quickly adapting to a new task. Unlike continual learning, meta-learning assumes a combined

(offline) access to meta-training tasks and does not concern itself with forgetting on previous tasks when adapting to new ones. Since the goal of faster adaptation is shared between meta and continual learning, recently some works have studied the two in conjunction [55, 62].

Curriculum learning. As opposed to learning from a randomly shuffled set of examples, curriculum learning learns a model from input examples organized in a meaningful order *e.g.* examples organized in the order of learning difficulty. Bengio et al. [20] showed that the order of examples affects both the convergence speed and the quality of the obtained minimum. Similar to continual learning, curriculum learning seeks positive forward and backward transfer, but unlike continual learning, it only considers a single task with offline access to the complete dataset of the task.

Online learning. Online learning [25, 58] refers to learning from a stream of data similar to continual learning. However, unlike continual learning, the considered stream belongs to a single task or domain. The main idea in online learning is to maintain a buffer of examples observed over a period of time and update the model at each time step such that the resultant model would incur the smallest loss compared to the best model obtained in hindsight on the buffer. Since only a single task is involved, the online learning literature does not concern itself with forgetting or negative backward transfer.

Causal Inference Informally, causal inference [104] refers to inferring the underlying causes of a phenomenon *e.g.* inferring the effects of any intervention or treatment *etc*. The main idea in causal inference is to recover the arrow of time describing the cause and effect, making the problem inherently sequential, similar to continual learning. While causal inference has extensively been studied in other

fields such as economics, epidemiology, public policy *etc*., it has recently seen a surge in machine learning [123, 65, 21]. More recently, Arjovsky et al. [10] proposed to learn invariant features that are stable across multiple environments (or tasks) as causal signals. Similar to multitask learning, causal inference recovers a shared structure that remains invariant across different tasks.

2.4 Continual learning formulation

We now describe the different continual learning setups that are used in this thesis.

Task. As noted previously, the main idea in continual learning is to learn from a sequence of *tasks*. We now concretize the notion of a task. A task in continual learning is a set of examples where either the input or the output distribution is fixed. In this thesis, we construct these tasks by manipulating the computer vision datasets as described in the next section. More specifically, we define these tasks by the corresponding datasets *i.e.* set $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$ will describe T tasks.

Training. A continual learner can experience these tasks in different ways. In a training setup where a continual learner can batch the complete dataset of a task, it can perform offline training on this dataset *i.e.*, repeating the training for multiple epochs before seeing the next task. Such training will be referred to as *multi-epoch* training in this thesis. On the other hand, when the data of a task arrives in a sequence, batch mode (offline) training is not possible. The model needs to learn after seeing an example only once. This training regime will be referred to as *single-epoch* training in this work. We note that single-epoch training is closer to the desiderata of continual learning described in section 2.2.

Evaluation. When a continual learner is evaluated, the data generating oracle can choose to release more information about a task with the help of a task descriptor *e.g.* in a multiclass classification task, it can choose to provide information about the correct model head to which a corresponding class belongs to. If the data generation process provides this information at the test time, the corresponding evaluation setting will be termed as *multihead*. In the absence of such information, the evaluation setting will be called *single-head*.

We note that the training and evaluation setups described above are based on the notion that the continual learning problem is divided into different tasks. There are other setups [5, 6] that do not rely on an explicit notion of a task and instead learn from the changes in the data distribution. These setups are called *task-free* setups. We will not focus on these setups in this thesis.

2.4.1 Benchmarks

We now provide a description of the datasets that are used to construct different continual learning scenarios in this thesis. The exact details of the total number of training samples, the number of training epochs, and the number of tasks for each dataset are provided in each chapter where they are used.

- **MNIST** [78]. Contains $70,000$ images of handwritten digits of 10 classes with balanced label distribution. Each image is of size 28×28. Two continual learning benchmarks are created from MNIST dataset. In *Permuted MNIST* benchmark, a task is defined by a certain random permutation of the input pixels that is applied to all images. Each permutation defines a new task. Similarly, in *Rotated MNIST* benchmark, a task is defined by applying a certain random rotation to all images. Different rotations define different tasks.

- **CIFAR-10** [68]. Contains $60,000$ colored images of 10 different objects with balanced label distribution. Each image is 32×32. *Split CIFAR-10* benchmark is created where the dataset is divided into 5 disjoint subsets, where each subset is constructed by randomly sampling 2 classes without replacement from a total of 10 classes. Each subset defines a task.

- **CIFAR-100** [68]. Contains $60,000$ colored images of 100 different objects with balanced label distribution. Each image is 32×32. *Split CIFAR-100* benchmark is created where the dataset is divided into 20 disjoint subsets, where each subset is constructed by randomly sampling 5 classes without replacement from a total of 100 classes. Each subset defines a task.

- **CUB** [147]. Contains 11788 colored images of 200 bird categories. Each image is 224×224. *Split CUB* benchmark is created where the dataset is divided into 20 disjoint subsets, where each subset is constructed by randomly sampling 10 classes without replacement from a total of 200 classes. Each subset defines a task.

- **AWA** [74]. Contains 37322 colored images of 50 animal categories. Each image is 224×224. *Split AWA* benchmark is created by sampling 5 classes with replacement from the total 50 classes, constructing 20 tasks. In this setting, classes may overlap among multiple tasks, but within each task they compete with different set of classes. Each subset defines a task.

- **miniImageNet** [145]. Contains $60,000$ colored images of 100 different objects with balanced label distribution. Each image is 84×84. *Split miniImageNet* benchmark is created where the dataset is divided into 20 disjoint subsets, where each subset is constructed by randomly sampling 5 classes without replacement from a total of 100 classes. Each subset defines a task.

2.4.2 Evaluation metrics

The desiderata of continual learning defined in section 2.2 require different, sometimes competing, evaluation metrics to measure performance on different axes. Below we provide an intuitive definition of the evaluation metrics used in this thesis. More formal definitions can be found in each chapter where they are used.

Accuracy. Accuracy measures the performance (percentage of correct predictions on a held out dataset) of each task. The final average accuracy measures the average performance of all tasks at the end of a continual learning experience *i.e.* when the model has seen the last task in the sequence. A continual learning model aims to achieve the highest possible accuracy. In all the scenarios considered in this thesis, the offline model trained with multitask error (cf. equation 2.1) is the upper bound on the average accuracy.

Forgetting. Forgetting measures the drop in the performance of each task, measured at the end of the training of all tasks, compared to the peak performance of the task throughout the continual learning experience. Note that the peak performance of a task does not need to be observed right after the training of the task. In this sense, forgetting actually measures the maximum forgetting of a task. The metric we report in this thesis, then, is the average maximum forgetting measured at the end of a continual learning experience. A continual learner tries to reduce the forgetting. A method that has simultaneous access to all tasks will have zero forgetting by construction.

Intransigence. Intransigence measures the inability of the model to learn a task. This metric is particularly useful for a model that constrains the learning of a new task to keep itself from forgetting previous tasks. This metric effectively captures the stability-plasticity dilemma described in section 1.2. A model that tries to

reduce the forgetting of previous tasks by limiting the learning of a new task will have higher intransigence. A continual learning algorithm aims to reduce the intransigence of a new task while preventing the forgetting of previous ones.

Learning Curve Area (LCA). Learning curve area quantifies the speed of learning of a continual learning algorithm. We aim for algorithms that learn new tasks quickly – with as few examples as possible. LCA measures the performance of a model after each mini-batch of training and integrates the performance for m mini-batches. The model which is transferring more information to new tasks will have a higher LCA for a given m. We report the average LCA across tasks for different values of m.

Chapter 3

Parameter Regularization-based Continual Learning: Riemannian Walk

Abstract

Continual learning (CL) has received a lot of attention recently, however, the literature lacks a precise problem definition, proper evaluation settings, and metrics tailored specifically for the CL problem. One of the main objectives of this chapter is to fill these gaps so as to provide a common ground for better understanding of CL. The main challenge for an CL algorithm is to update the classifier whilst preserving existing knowledge. We observe that, in addition to *forgetting*, a known issue while preserving knowledge, CL also suffers from a problem we call *intransigence*, inability of a model to update its knowledge. We introduce two metrics to quantify *forgetting* and *intransigence* that allow us to understand, analyse, and gain better insights into the behaviour of CL algorithms. We present RWalk, a generalization of EWC++ (our ef-

ficient version of EWC [67]) and Path Integral [156] with a theoretically grounded KL-divergence based perspective. We provide a thorough analysis of various CL algorithms on MNIST and CIFAR-100 datasets. In these experiments, RWalk obtains superior results in terms of accuracy, and also provides a better trade-off between forgetting and intransigence.

3.1 Introduction

Realizing human-level intelligence requires developing systems capable of learning new tasks continually while preserving *knowledge* about the old ones. This is precisely the objective underlying continual learning (CL) algorithms. By definition, CL has ever-expanding output space, and limited or no access to data from the previous tasks while learning a new one. This makes CL more challenging and fundamentally different from the classical supervised learning paradigm where the output space is fixed and the entire dataset is available. Recently, there have been several works in CL [67, 87, 110, 156] with varying evaluation settings and metrics making it difficult to establish fair comparisons. The first objective of this chapter is to rectify these issues by providing precise definitions, evaluation settings, and metrics for CL for the classification task.

Let us now discuss the key points to consider while designing CL algorithms. The first question is *'how to define knowledge: factors that quantify what the model has learned'*. Usually, knowledge is defined either using the input-output behaviour of the network [57, 110] or the network parameters [67, 156]. Once the knowledge is defined, the objective then is to *preserve* and *update* it to counteract two inherent issues with CL algorithms: (1) *forgetting*: catastrophically forgetting knowledge of previous tasks; and (2) *intransigence*: inability to update the knowledge to learn

a new task. Both of these problems require contradicting solutions and pose a trade-off for any CL algorithm.

To capture this trade-off, we advocate the use of measures that evaluate a CL algorithm based on its performance on the *past* and the *present* tasks in the hope that this will reflect in the algorithm's behaviour on the *future* unseen tasks. Taking this into account we introduce two metrics to evaluate *forgetting* and *intransigence*. These metrics together with the standard multi-class average accuracy allow us to understand, analyse, and gain better insights into the behaviour of various CL algorithms.

Further, we present a generalization of two recently proposed continual learning algorithms, Elastic Weight Consolidation (EWC) [67], and Path Integral (PI) [156]. In particular, first we show that in EWC, while learning a new task, the model's likelihood distribution is regularized using a well known second-order approximation of the KL-divergence [7, 102], which is equivalent to computing distance in a Riemannian manifold induced by the Fisher Information Matrix [7]. To compute and update the Fisher information matrix, we use an efficient (in terms of memory) and online (in terms of computation) approach, leading to a faster and online version of EWC which we call EWC++. Note that, a similar extension to EWC, called online-EWC, is concurrently proposed by Schwarz *et al*. [127]. Next, we modify the PI [156] where instead of computing the change in the loss per unit distance in the Euclidean space between the parameters as the measure of sensitivity, we use the approximate KL divergence (distance in the Riemannian manifold) between the output distributions as the distance to compute the sensitivity. This gives us the *parameter importance score* which is accumulated over the optimization trajectory effectively encoding the information about all the seen tasks so far. Finally, RWalk is obtained by combining EWC++ and the modified PI.

Furthermore, in order to counteract *intransigence*, we study different *sampling*

strategies that store a small representative subset ($\leq 5\%$) of the dataset from the previous tasks. This not only allows the network to *recall* information about the previous tasks but also helps in learning to *discriminate* current and previous tasks. Finally, we present a thorough analysis to better understand the behaviour of CL algorithms on MNIST [78] and CIFAR-100 [68] datasets. To summarize, our main contributions in this chapter are:

1. New evaluation metrics - *Forgetting* and *Intransigence* - to better understand the behaviour and performance of a continual learning algorithm.
2. EWC++: An efficient and online version of EWC.
3. RWalk: A generalization of EWC++ and PI with theoretically grounded KL-divergence based perspective providing new insights.
4. Analysis of different methods in terms of accuracy, forgetting, and intransigence.

3.2 Problem Set-up and Preliminaries

Here we define the CL problem and discuss the practicality of two different evaluation settings: (a) single-head; and (b) multi-head. In addition, we review the probabilistic interpretation of neural networks and the connection of KL-divergence with the distance in the Riemannian manifold, both of which are crucial to our approach.

3.2.1 Single-head vs Multi-head Evaluations

We consider a stream of tasks, each corresponding to a set of labels. For the k-th task, let $\mathcal{D}_k = \{(\mathbf{x}_i^k, y_i^k)\}_{i=1}^{n_k}$ be the dataset, where $\mathbf{x}_i^k \in \mathcal{X}$ is the input and $y_i^k \in \mathbf{y}^k$ the ground truth label, and $\mathbf{y}^k$ is the set of labels specific to the task.

The main distinction between the single-head and the multi-head evaluations is that, at test time, in *single-head*, the task identifier (k) is unknown, whereas in *multi-head*, it is given. Therefore, for the single-head evaluation, the objective at the k-th task is to learn a function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}^k$, where $\mathcal{Y}^k = \cup_{j=1}^{k} \mathbf{y}^j$ corresponds to all the known labels. For multi-head, as the task identifier is known, $\mathcal{Y}^k = \mathbf{y}^k$. For example, consider MNIST with 5 tasks: $\{\{0,1\}, \cdots, \{8,9\}\}$; trained in an incremental manner. Then, at the 5-th task, for a given image, the *multi-head* evaluation is to predict a class out of two labels $\{8,9\}$ for which the 5-th task was trained. However, the *single-head* evaluation at 5-th task is to predict a label out of all the ten classes $\{0, \cdots, 9\}$ that the model has seen thus far.

Why single-head evaluation for CL?

In the case of *single-head*, used by [82, 110], the output space consists of all the known labels. This requires the classifier to learn to distinguish labels from different tasks as well. Since, the tasks are supplied in a sequence in CL, while learning a new task, the classifier must also be able to learn the inter-task discrimination with no or limited access[1] to the previous tasks' data. This is a much harder problem compared to multi-head where the output space contains labels of the current task only. Furthermore, single-head is a more practical setting because knowing the subset of labels to look at a priori, which is the case in multi-head, requires extra supervisory signals, in the form of task descriptors, at test time that reduce the problem complexity.

[1]Since the number of tasks are potentially unlimited in CL, it is impossible to store all the previous data in a scalable manner.

3.2.2 Probabilistic Interpretation of Neural Network Output

If the final layer of a neural network is a soft-max layer and the network is trained using cross entropy loss, then the output may be interpreted as a probability distribution over the categorical variables. Thus, at a given θ, the conditional likelihood distribution learned by a neural network is actually a conditional multinoulli distribution defined as $p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{K} p_{\theta,j}^{[y=j]}$, where $p_{\theta,j}$ is the soft-max probability of the j-th class, K are the total number of classes, $\mathbf{y}$ is the one-hot encoding of length K, and $[\cdot]$ is Iverson bracket. A prediction can then be obtained from the likelihood distribution $p_\theta(\mathbf{y}|\mathbf{x})$. Typically, instead of sampling, a label with the highest soft-max probability is chosen as the network's prediction. Note that, if $\mathbf{y}$ corresponds to the one-hot ground-truth label then the log-likelihood is exactly the same as the negative of the cross-entropy loss, *i.e.*, if the ground-truth corresponds to the t-th index of the one-hot representation of $\mathbf{y}$, then $\log p_\theta(\mathbf{y}|\mathbf{x}) = \log p_{\theta,t}$ (more details can be found in the appendix).

3.2.3 KL-divergence as the Distance in the Riemannian Manifold

Let $D_{\mathrm{KL}}(p_\theta \| p_{\theta+\Delta\theta})$ be the KL-divergence [71] between the conditional likelihoods of a neural network at θ and $\theta + \Delta\theta$, respectively. Then, assuming $\Delta\theta \to 0$, the second-order Taylor approximation of the KL-divergence can be written as $D_{\mathrm{KL}}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta = \frac{1}{2}\|\Delta\theta\|_{F_\theta}^2$ [1], where F_θ, known as the *empirical Fisher Information Matrix* [7, 102] at θ, is defined as:

$$F_\theta = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\left(\frac{\partial \log p_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta}\right)\left(\frac{\partial \log p_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta}\right)^\top\right], \tag{3.1}$$

where $\mathcal{D}$ is the dataset. Note that, as mentioned earlier, the log-likelihood $\log p_\theta(\mathbf{y}|\mathbf{x})$ is the same as the negative of the cross-entropy loss function, thus, F_θ can be seen

[1] Proof and insights are provided in the appendix.

as the *expected loss-gradient covariance matrix*. By construction (outer product of gradients), F_θ is positive semi-definite (PSD) which makes it highly attractive for second-order optimization techniques [7, 102, 50, 77, 90]. When $\Delta\theta \to 0$, computing KL-divergence $\frac{1}{2}\|\Delta\theta\|^2_{F_\theta}$ is equivalent to computing the *distance* in a Riemannian manifold[1] [81] induced by the Fisher information matrix at θ. Since $F_\theta \in \mathbb{R}^{P\times P}$ and P is usually in the order of millions for neural networks, it is practically infeasible to store F_θ. To handle this, similar to [67], we assume parameters to be independent of each other (diagonal F_θ) which results in the following approximation of the KL-divergence:

$$D_{\mathrm{KL}}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\sum_{i=1}^{P} F_{\theta_i}\,\Delta\theta_i^2\ , \tag{3.2}$$

where θ_i is the i-th entry of θ. Notice, the diagonal entries of F_θ are the expected square of the gradients, where the expectation is over the entire dataset. This makes F_θ computation expensive as it requires a full forward-backward pass over the dataset.

3.3 Forgetting and Intransigence

Since the objective is to continually learn new tasks while preserving knowledge about the previous ones, a CL algorithm should be evaluated based on its performance both on the *past* and the *present* tasks in the hope that this will reflect in algorithm's behaviour on the *future* unseen tasks. To achieve this, along with average accuracy, there are two crucial components that must be quantified (1) *forgetting*: how much an algorithm forgets what it learned in the past; and (2) *intransigence*: inability of an algorithm to learn new tasks. Intuitively, if a model is heavily regularized over previous tasks to preserve knowledge, it will forget less but have high intransigence. If, in contrast, the regularization is too weak, while

[1]Since F_θ is PSD, this makes it a pseudo-manifold.

the intransigence will be small, the model will suffer from catastrophic forgetting. Ideally, we want a model that suffers less from both, thus efficiently utilizing a finite model capacity. In contrast, if one observes high negative correlation between forgetting and intransigence, which is usually the case, then, it suggests that either the model capacity is saturated or the method does not effectively utilize the capacity. Before defining metrics for quantifying forgetting and intransigence, we first define the multi-class average accuracy which will be the basis for defining the other two metrics. Note, some other task specific measure of correctness (*e.g.*, IoU for object segmentation) can also be used while the definitions of forgetting and intransigence remain the same.

Average Accuracy (A)

Let $a_{k,j} \in [0, 1]$ be the accuracy (fraction of correctly classified images) evaluated on the held-out test set of the j-th task $(j \leq k)$ after training the network incrementally from tasks 1 to k. Note that, to compute $a_{k,j}$, the output space consists of either $\mathbf{y}^j$ or $\cup_{j=1}^{k}\mathbf{y}^j$ depending on whether the evaluation is multi-head or single-head (refer section 3.2.1). The average accuracy at task k is then defined as $A_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j}$. The higher the A_k the better the classifier, but this does not provide any information about forgetting or intransigence profile of the CL algorithm which would be crucial to judge its behaviour.

Forgetting Measure (F)

We define forgetting for a particular task (or label) as the difference between the *maximum* knowledge gained about the task throughout the learning process in the past and the knowledge the model currently has about it. This, in turn, gives an estimate of how much the model forgot about the task given its current state. Following this, for a classification problem, we quantify forgetting for the j-th task

after the model has been incrementally trained up to task $k > j$ as:

$$f_j^k = \max_{l \in \{1,\cdots,k-1\}} a_{l,j} - a_{k,j} \ , \quad \forall j < k \ . \tag{3.3}$$

Note, $f_j^k \in [-1, 1]$ is defined for $j < k$ as we are interested in quantifying forgetting for *previous* tasks. Moreover, by normalizing against the number of tasks seen previously, the average forgetting at k-th task is written as $F_k = \frac{1}{k-1} \sum_{j=1}^{k-1} f_j^k$. Lower F_k implies less forgetting on previous tasks. Here, instead of $\max$ one could use *expectation* or $a_{j,j}$ in order to quantify the knowledge about a task in the past. However, taking $\max$ allows us to estimate forgetting along the learning process as explained below.

Positive/Negative Backward Transfer ((P/N)BT): Backward transfer (BT) is defined in [87] as the influence that learning a task k has on the performance of a previous task $j < k$. Since our objective is to measure forgetting, negative forgetting ($f_j^k < 0$) implies positive influence on the previous task or positive backward transfer (PBT), the opposite for NBT. Furthermore, in [87], $a_{j,j}$ is used in place of $\max_{l \in \{1,\cdots,k-1\}} a_{l,j}$ (refer equation 3.3) which makes the measure agnostic to the CL process and does not effectively capture forgetting. To understand this, let us consider an example with 4 tasks trained in an incremental manner. We are interested in measuring forgetting of task 1 after training up to task 4. Let the accuracies be $\{a_{1,1}, a_{2,1}, a_{3,1}, a_{4,1}\} = \{0.7, 0.8, 0.6, 0.5\}$. Here, forgetting measured based on equation 3.3 is $f_1^4 = 0.3$, whereas [87] would measure it as 0.2 (irrespective of the variations in $a_{2,1}$ and $a_{3,1}$). Hence, it does not capture the fact that there was a PBT in the learning process. We believe, it is vital that an evaluation metric of an CL algorithm considers such behaviour along the learning process.

Intransigence Measure (I)

We define *intransigence* as the inability of a model to learn new tasks. The effect of intransigence is more prominent in the single-head setting especially in the absence of previous data, as the model is expected to learn to differentiate the current task from the previous ones. Experimentally we show that storing just a few representative samples (refer section 3.4.2) from the previous tasks improves intransigence significantly. Since we wish to quantify the *inability* to learn, we compare to the standard classification model which has access to all the datasets at all times. We train a reference/target model with dataset $\bigcup_{l=1}^{k} \mathcal{D}_l$ and measure its accuracy on the held-out set of the k-th task, denoted as a_k^*. We then define the intransigence for the k-th task as:

$$I_k = a_k^* - a_{k,k} \ , \tag{3.4}$$

where $a_{k,k}$ denotes the accuracy on the k-th task when trained up to task k in an incremental manner. Note, $I_k \in [-1, 1]$, and lower the I_k the better the model. A reasonable reference/target model can be defined depending on the feasibility to obtain it. In situations where it is highly expensive, an approximation can be proposed.

Positive/Negative Forward Transfer ((P/N)FT): Since intransigence is defined as the gap between the accuracy of a CL algorithm and the reference model, negative intransigence ($I_k < 0$) implies learning incrementally up to task k positively influences model's knowledge about it, *i.e.*, positive forward transfer (PFT). Similarly, $I_k > 0$ implies NFT. However, in [87], FT is quantified as the gain in accuracy compared to the random guess (not a measure of intransigence) which is complementary to our approach.

3.4 Riemannian Walk for continual learning

We first describe EWC++, an efficient version of the well known EWC [67], and then RWalk which is a generalization of EWC++ and PI [156]. Briefly, RWalk has three key components: (1) a KL-divergence-based regularization over the conditional likelihood $p_\theta(\mathbf{y}|\mathbf{x})$ (EWC++); (2) a parameter importance score based on the sensitivity of the loss over the movement on the Riemannian manifold (similar to PI); and (3) strategies to obtain a few representative samples from the previous tasks. The first two components mitigate the effects of catastrophic forgetting, whereas the third handles intransigence.

3.4.1 Avoiding Catastrophic Forgetting

KL-divergence based Regularization (EWC++)

We learn parameters for the current task such that the new conditional likelihood is close (in terms of KL) to the one learned until previous tasks. To achieve this, we regularize over the conditional likelihood distributions $p_\theta(\mathbf{y}|\mathbf{x})$ using the approximate KL-divergence, equation 3.2, as the distance measure. This regularization would preserve the inherent properties of the model about previous tasks as the learning progresses. Thus, given parameters θ^{k-1} trained sequentially from task 1 to $k-1$, and dataset $\mathcal{D}_k$ for the k-th task, the objective is:

$$\arg\min_{\theta} \tilde{L}^k(\theta) := L^k(\theta) + \lambda\, D_{\mathrm{KL}}\left(p_{\theta^{k-1}}(\mathbf{y}|\mathbf{x}) \| p_\theta(\mathbf{y}|\mathbf{x})\right) \ , \tag{3.5}$$

where, λ is a hyperparameter. Substituting equation 3.2, the KL-divergence component can be written as $D_{\mathrm{KL}}\left(p_{\theta^{k-1}} \| p_\theta\right) \approx \frac{1}{2}\sum_{i=1}^{P} F_{\theta_i^{k-1}}(\theta_i - \theta_i^{k-1})^2$. Note that, for two tasks, the above regularization is exactly the same as that of EWC [67]. Here we presented it from the KL-divergence based perspective. Another way to look at

it would be to consider the Fisher[1] for each parameter to be its importance score. The intuitive explanation for this is as follows; since the Fisher captures the local curvature of the KL-divergence surface of the likelihood distribution (as it is the component of the second-order term of Taylor approximation, refer section 3.2.3), higher Fisher implies higher curvature, thus suggests to move less in that direction in order to preserve the likelihood.

In the case of multiple tasks, EWC requires storing the Fisher for each task independently ($\mathcal{O}(kP)$ parameters), and regularizing over all of them jointly. This is practically infeasible if there are many tasks and the network has millions of parameters. Moreover, to estimate the empirical Fisher, EWC requires an additional pass at the end of training over the dataset of each task (see equation 3.1). To address these two issues, we propose EWC++ that (1) maintains single diagonal Fisher information matrix as the training over tasks progresses, and (2) uses moving average for its efficient update similar to [90]. Given F_θ^{t-1} at $t-1$, Fisher in EWC++ is updated as:

$$F_\theta^t = F_\theta^{t-1} + \alpha(F_\theta^t - F_\theta^{t-1}), \tag{3.6}$$

where F_θ^t is the Fisher matrix obtained using the *current batch* and $\alpha \in [0, 1]$ is a hyperparameter. Note, t represents the training iterations, thus, computing Fisher in this manner contains information about previous tasks, and also eliminates the additional forward-backward pass over the dataset. At the end of each task, we simply store F_θ^t as $F_{\theta^{k-1}}$ and use it to regularize the next task, thus storing only two Fisher information matrices at any instant during training, irrespective of the number of tasks. Similar to EWC++, an efficient version of EWC, referred as online-EWC, is concurrently developed in [127].

In EWC, Fisher is computed at a local minimum of $\tilde{L}^k$ using the gradients of L^k, which is nearly zero whenever $\tilde{L}^k \approx L^k$ (*e.g.*, smaller λ or when $k = 1$). This

[1]By Fisher we always mean the empirical Fisher information matrix.

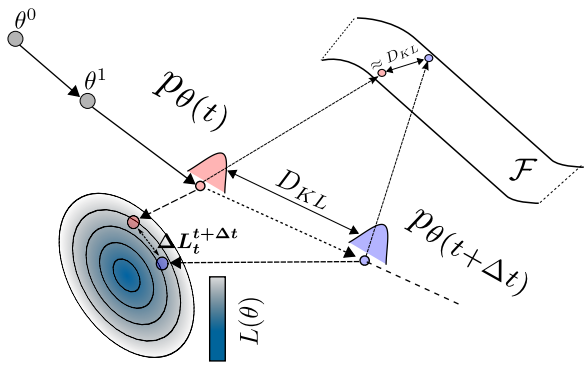


Figure 3.1: Parameter importance accumulated over the optimization trajectory.

results in negligible regularization leading to catastrophic forgetting. This issue is partially addressed in EWC++ using moving average. However, to improve it further and to capture model's behaviour not just at the minimum but also during the entire training process, we augment each element of the diagonal Fisher with a positive scalar as described below. This also ensures that the augmented Fisher is always positive-definite.

Optimization-path based Parameter Importance

Since Fisher captures the intrinsic properties of the model and it only depends on L^k, it is blinded towards the influence of parameters over the optimization path on the loss surface of $\tilde{L}^k$. We augment Fisher with a parameter importance score which is accumulated over the entire training trajectory of $\tilde{L}^k$ (similar to [156]). This score is defined as the ratio of the change in the loss function to the distance between the conditional likelihood distributions per step in the parameter space.

More precisely, for a change of parameter from $\theta_i(t)$ to $\theta_i(t+1)$ (where t is the time step or training iteration), we define parameter importance as the ratio of the change in the loss to its influence in $D_{\mathrm{KL}}(p_{\theta(t)} \| p_{\theta(t+1)})$. Intuitively, importance will be higher if a small change in the distribution causes large improvement over the loss. Formally, using the first-order Taylor approximation, the change in loss L can

be written as:

$$L(\theta(t+\Delta t)) - L(\theta(t)) \approx -\sum_{i=1}^{P}\sum_{t=t}^{t+\Delta t} \frac{\partial L}{\partial \theta_i}\left(\theta_i(t+1) - \theta_i(t)\right) = -\sum_{i=1}^{P} \Delta L_t^{t+\Delta t}(\theta_i), \quad (3.7)$$

where $\frac{\partial L}{\partial \theta_i}$ is the gradient at t, and $\Delta L_t^{t+\Delta t}(\theta_i)$ represents the accumulated change in the loss caused by the change in the parameter θ_i from time step t to $t+\Delta t$. This change in parameter would cause a corresponding change in the model distribution which can be computed using the approximate KL-divergence (equation 3.2). Thus, the importance of the parameter θ_i from training iteration t_1 to t_2 can be computed as $s_{t_1}^{t_2}(\theta_i) = \sum_{t=t_1}^{t_2} \frac{\Delta L_t^{t+\Delta t}(\theta_i)}{\frac{1}{2}F_{\theta_i}^t \Delta\theta_i(t)^2 + \epsilon}$, where $\Delta\theta_i(t) = \theta_i(t+\Delta t) - \theta_i(t)$ and $\epsilon > 0$. The denominator is computed at every discrete intervals of $\Delta t \geq 1$ and $F_{\theta_i}^t$ is computed efficiently at every t-th step using moving average as described while explaining EWC++. The computation of this importance score is illustrated in figure 3.1. Since we care about the positive influence of the parameters, negative scores are set to zero. Note that, if the Euclidean distance is used instead, the score $s_{t_1}^{t_2}(\theta_i)$ would be similar to that of PI [156].

Final Objective Function (RWalk)

We now combine Fisher information matrix based importance and the optimization-path based importance scores as follows:

$$\tilde{L}^k(\theta) = L^k(\theta) + \lambda \sum_{i=1}^{P} (F_{\theta_i^{k-1}} + s_{t_0}^{t_{k-1}}(\theta_i))(\theta_i - \theta_i^{k-1})^2 \,. \quad (3.8)$$

Here, $s_{t_0}^{t_{k-1}}(\theta_i)$ is the score accumulated from the first training iteration t_0 until the last training iteration t_{k-1}, corresponding to task $k-1$. Since the scores are accumulated over time, the regularization gets increasingly rigid. To alleviate this and enable continual learning, after each task the scores are averaged: $s_{t_0}^{t_{k-1}}(\theta_i) =$

$\frac{1}{2}\left(s_{t_0}^{t_{k-2}}(\theta_i) + s_{t_{k-2}}^{t_{k-1}}(\theta_i)\right)$. This continual averaging makes the tasks learned far in the past less influential than the tasks learned recently. Furthermore, while adding, it is important to make sure that the scales of both $F_{\theta_i^{k-1}}$ and $s_{t_0}^{t_{k-1}}(\theta_i)$ are in the same order, so that the influence of both the terms is retained. This can be ensured by individually normalizing them to be in the interval $[0, 1]$. *This, together with score averaging, have a positive side-effect of the regularization hyperparameter λ being less sensitive to the number of tasks.* Whereas, EWC [67] and PI [156] are highly sensitive to λ, making them relatively less reliable for CL. Note, during training, the space complexity for RWalk is $\mathcal{O}(P)$, independent of the number of tasks.

3.4.2 Handling Intransigence

Experimentally, we observed that training k-th task with $\mathcal{D}_k$ leads to a poor test accuracy for the current task compared to previous tasks in the *single-head* evaluation setting (refer section 3.2.1). This happens because during training the model has access to $\mathcal{D}_k$ which contains labels only for the k-th task, $\mathbf{y}^k$. However, at test time the label space is over all the tasks seen so far $\mathcal{Y}^k = \cup_{j=1}^{k}\mathbf{y}^j$, which is much larger than $\mathbf{y}^k$. This in turn increases *confusion* at test time as the predictor function has no means to differentiate the samples of the current task from the ones of previous tasks. An intuitive solution to this problem is to store a small subset of representative samples from the previous tasks and use it while training the current task [110]. Below we discuss different strategies to obtain such a subset. Note that we store m points from each task-specific dataset as the training progresses, however, it is trivial to have a fixed total number of samples for all the tasks similar to iCaRL [110].

Uniform Sampling

A naïve yet highly effective (shown experimentally) approach is to sample uniformly at random from the previous datasets.

Plane Distance-based Sampling

In this case, we assume that samples closer to the decision boundary are more representative than the ones far away. For a given sample $\{\mathbf{x}_i, y_i\}$, we compute the pseudo-distance from the decision boundary $d(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top w^{y_i}$, where $\phi(\cdot)$ is the feature mapping learned by the neural network and w^{y_i} are the last fully connected layer parameters for class y_i. Then, we sample points based on $q(\mathbf{x}_i) \propto \frac{1}{d(\mathbf{x}_i)}$. Here, the intuition is, since the change in parameters is regularized, the feature space and the decision boundaries do not vary much. Hence, the samples that lie close to the boundary would act as *boundary defining samples*.

Entropy-based Sampling

Given a sample, the entropy of the output soft-max distribution measures the uncertainty of the sample which we used to sample points. The higher the entropy the more likely is that the sample would be picked.

Mean of Features (MoF)

iCaRL [110] proposes a method to find samples based on the feature space $\phi(\cdot)$. For each class y, m number of points are found whose mean in the feature space closely approximate the mean of the entire dataset for that class. However, this subset selection strategy is inefficient compared to the above sampling methods. In fact, the time complexity is $\mathcal{O}(nfm)$ where n is dataset size, f is the feature dimension and m is the number of required samples.

3.5 Related Work

One way to address catastrophic forgetting is by dynamically expanding the network for each new task [80, 109, 118, 139]. Though intuitive and simple, these approaches are not scalable as the size of the network increases with the number of tasks. A better strategy would be to exploit the over-parametrization of neural networks [56]. This entails regularizing either over the activations (output) [110, 85] or over the network parameters [67, 156]. Even though activation-based approach allows more flexibility in parameter updates, it is memory inefficient if the activations are in millions, *e.g.*, semantic segmentation. On the contrary, methods that regularize over the parameters - weighting the parameters based on their individual *importance* - are suitable for such tasks. Our method falls under the latter category and we show that our method is a generalization of EWC++ and PI [156], where EWC++ is our efficient version of EWC [67], very similar to the concurrently developed online-EWC [127]. Similar in spirit to regularization over the parameters, Lee et al. [82] use moment matching to obtain network weights as the combination of the weights of all the tasks, and Nguyen et al. [98] enforce the distribution over the model parameters to be close via a Bayesian framework. Different from the above approaches, Lopez-Paz and Ranzato [87] update gradients such that the losses of the previous tasks do not increase, while Shin et al. [129] resort to a retraining strategy where the samples of the previous tasks are generated using a learned generative model.

3.6 Experiments

Datasets

We evaluate baselines and our proposed model - RWalk - on two datasets:

1. *Incremental MNIST*: The standard MNIST dataset is split into five disjoint subsets (tasks) of two consecutive digits, *i.e.*, $\cup_k \mathbf{y}^k = \{\{0,1\},\ldots,\{8,9\}\}$.

2. *Incremental CIFAR*: To show that our approach scales to bigger datasets, we use incremental CIFAR where CIFAR-100 dataset is split into ten disjoint subsets such that $\cup_k \mathbf{y}^k = \{\{0-9\},\ldots,\{90-99\}\}$.

Architectures

The architectures used are similar to [156]. For MNIST, we use an MLP with two hidden layers each having 256 units with ReLU nonlinearites. For CIFAR-100, we use a CNN with four convolutional layers followed by a single dense layer (see appendix for more details). In all experiments, we use Adam optimizer [66] (learning rate = 1×10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$) with a fixed batch size of 64.

Baselines

We compare RWalk against the following baselines:

- Vanilla: Network trained without any regularization over past tasks.
- EWC [67] and PI [156]: Both use parameter based regularization. Note, we observed that EWC++ performed at least as good as EWC and therefore, in all the experiments, by EWC we mean the more efficient EWC++.
- iCaRL [110]: Uses regularization over the activations and a nearest-exemplar-based classifier. Here, iCaRL-hb1 refers to the *hybrid1* version, which uses the standard neural network classifier. Both the versions use previous samples.

Note, we use a few samples from the previous tasks to consolidate our baselines further in the single-head setting.

Table 3.1: Comparison with different baselines on MNIST and CIFAR in both multi-head and single-head evaluation settings. Baselines where samples are used are appended with '-S'. For MNIST and CIFAR, 10 (0.2%) and 25(5%) samples are used from the previous tasks using mean of features (MoF) based sampling strategy (refer section 3.4.2).

Methods	**MNIST**				**CIFAR**			
	Multi-head Evaluation							
	λ	$A_5(\%)$	F_5	I_5	λ	$A_{10}(\%)$	F_{10}	I_{10}
Vanilla	0	90.3	0.12	6.6×10^{-4}	0	44.4	0.36	0.02
EWC	75000	**99.3**	0.001	0.01	3×10^6	72.8	0.001	0.07
PI	0.1	**99.3**	0.002	0.01	10	73.2	0	0.06
RWalk (Ours)	1000	**99.3**	0.003	0.01	1000	**74.2**	0.004	0.04
	Single-head Evaluation							
Vanilla	0	38.0	0.62	0.29	0	10.2	0.36	-0.06
EWC	75000	55.8	0.08	0.77	3×10^6	23.1	0.03	0.17
PI	0.1	57.6	0.11	0.8	10	22.8	0.04	0.2
iCaRL-hb1	-	36.6	0.68	-0.01	-	7.4	0.40	0.06
iCaRL	-	55.8	0.19	0.46	-	9.5	0.11	0.35
Vanilla-S	0	73.7	0.30	0.03	0	12.9	0.64	-0.3
EWC-S	75000	79.7	0.14	0.22	15×10^5	33.6	0.27	-0.05
PI-S	0.1	78.7	0.24	0.05	10	33.6	0.27	-0.03
RWalk (Ours)	1000	**82.5**	0.15	0.14	500	**34.0**	0.28	-0.06

Results

We report the results in table 3.1 where RWalk outperforms all the baselines in terms of average accuracy and provides better trade-off between forgetting and intransigence. We now discuss the results in detail.

In the multi-head evaluation setting [156, 87], except Vanilla, all the methods provide state-of-the-art accuracy with *almost zero* forgetting and intransigence (top row of figure 3.2). *This gives an impression that CL problem is solved.* However, as discussed in section 3.2.1, this is an easier evaluation setting and does not capture the essence of CL.

However, in the single-head evaluation, *forgetting* and *intransigence* increase substantially due to the inability of the network to differentiate among tasks. Hence,

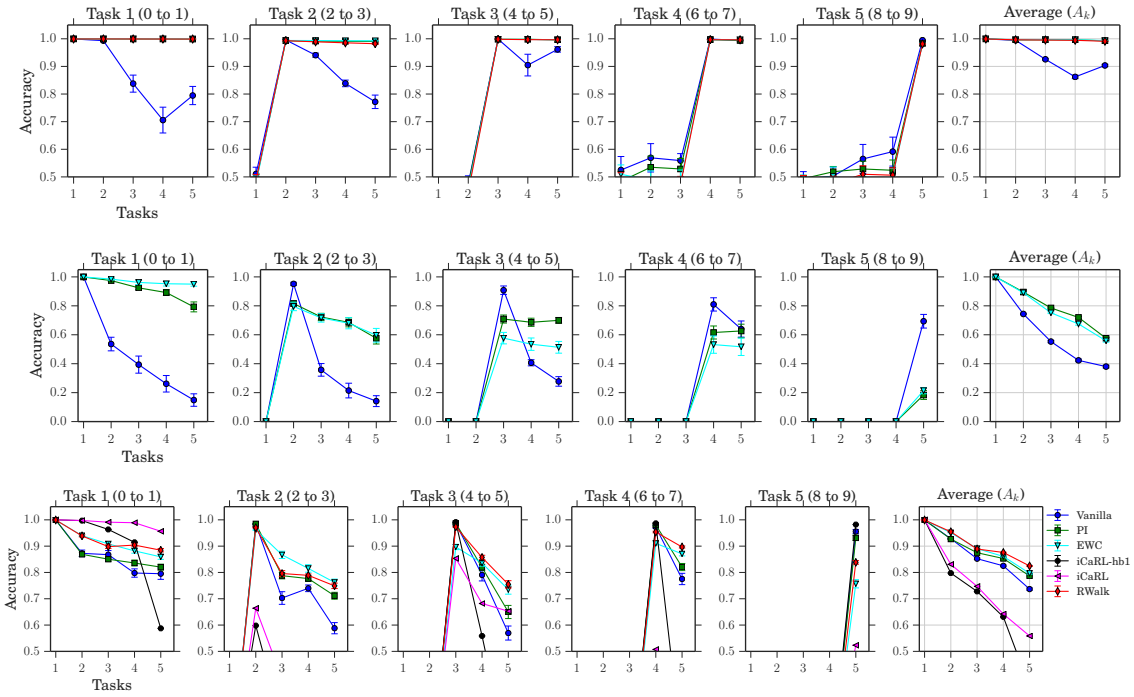
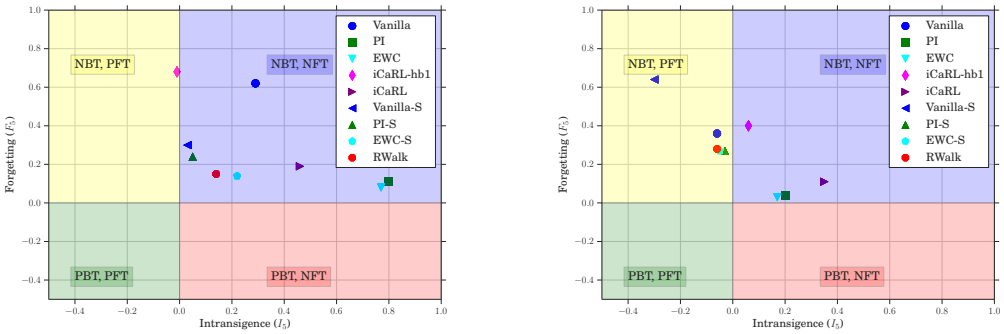


Figure 3.2: Accuracy on incremental MNIST with multi-head evaluation (top), and single-head evaluation without (middle) and with samples (bottom). First five columns show the variation in performance for different tasks, *e.g.*, the first plot depicts the performance variation on Task 1 when trained incrementally over five tasks. The last column shows the accuracy (A_k, refer section 3.3). Mean of features (MoF) sampling is used.

the performance significantly drops for all the methods (refer table 3.1 and the middle row of figure 3.2). For instance, on MNIST, forgetting and intransigence of Vanilla deteriorates from 0.12 to 0.62, and 6.6×10^{-4} to 0.29, respectively, causing the average accuracy to drop from 90.3% to 38.0%. Although, regularized methods, EWC and PI, designed to counter catastrophic forgetting, result in less degradation of forgetting, their accuracy is still significantly worse - compare 99.3% of PI in multi-head against 57.6% in single-head. In table 3.1, a similar performance decrease is observed on CIFAR-100 as well. Such a degradation in accuracy even with less forgetting shows that it is not only important to preserve knowledge (quantified by forgetting) but also to update knowledge (captured by intransigence) to achieve better performance. Task-level analysis for CIFAR dataset, similar to figure 3.2, is



(a) MNIST (b) CIFAR

Figure 3.3: Interplay between forgetting and intransigence.

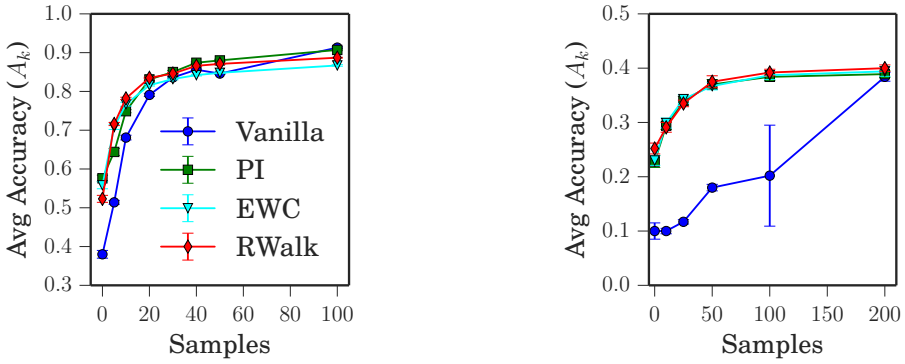
presented in the appendix.

We now show that even with a few representative samples intransigence can be mitigated. For example, in the case of PI on MNIST with only 10 ($\approx 0.2\%$) samples for each previous class, the intransigence drops from 0.8 to 0.05 which results in improving the average accuracy from 57.6% to 78.7%. Similar improvements can be seen for other methods as well. On CIFAR-100, with only 5% representative samples, almost identical behaviour is observed.

In our CIFAR-100 experiments (CNN instead of ResNet32), we note that the performance of iCaRL [110] is significantly worse than what has been reported by the authors. We believe this is due to the dependence of iCaRL on a highly expressive feature space, as both the regularization and the classifier depend on it. Perhaps, this reduced expressivity of the feature space due to the smaller network resulted in the performance loss.

Interplay of Forgetting and Intransigence

In figure 3.3 we study the interplay of forgetting and intransigence in the single-head setting. Ideally we would like a model to be in the quadrant marked as *PBT, PFT* (*i.e.*, positive backward transfer and positive forward transfer). On MNIST,



(a) MNIST

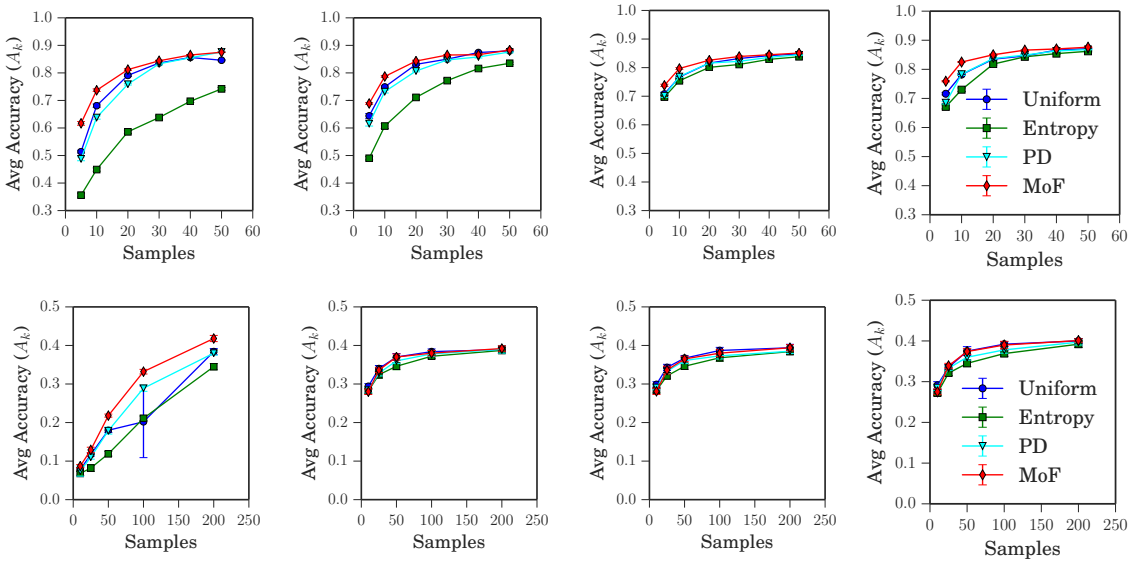
(b) CIFAR

Figure 3.4: Comparison by increasing the number of samples. On MNIST and CIFAR each class has around 5000 and 500 samples, respectively. With increasing number of samples, the performance of Vanilla improved, but in the range where Vanilla is poor, RWalk consistently performs the best. Uniform sampling is used.

since all the methods, except iCaRL-hb1, lie on the top-right quadrant, hence for models with comparable accuracy, a model which has the smallest distance from $(0, 0)$ would be better. As evident, RWalk is closest to $(0, 0)$, providing a better trade-off between forgetting and intransigence compared to all the other methods. On CIFAR-100, the models lie on both the top quadrants and with the introduction of samples, all the regularized methods show positive forward transfer. Since the models lie on different quadrants, their comparison of forgetting and intransigence becomes application specific. In some cases, we might prefer a model that performs well on new tasks (better intransigence), irrespective of its performance on the old ones (can compromise forgetting), and vice versa. Note that, RWalk maintains comparable performance to other baselines while yielding higher average accuracy on CIFAR-100.

Effect of Increasing the Number of Samples

As expected, for smaller number of samples, regularized methods perform far superior compared to Vanilla (refer figure 3.4). However, once the number of



(a) Vanilla (b) PI (c) EWC (d) RWalk

Figure 3.5: Comparison of different sampling strategies discussed in section 3.4.2 on MNIST (top) and CIFAR-100 (bottom). Mean of features (MoF) outperforms others.

samples are sufficiently large, Vanilla starts to perform better or equivalent to the regularized models. The reason is simple because now the Vanilla has access to enough samples of the previous tasks to relearn them at each step, thereby obviating the need of regularized models. However, in a CL problem, a fixed small-sized memory budget is usually assumed. Therefore, one cannot afford to store large number of samples from previous tasks. Additionally, for a simpler dataset like MNIST, Vanilla quickly catches up to the regularized models with small number of samples (20, 0.4% of total samples) but on a more challenging dataset like CIFAR it takes considerable amount of samples (200, 40% of total samples) of previous tasks for Vanilla to match the performance of the regularized models.

Comparison of Different Sampling Strategies

In figure 3.5 we compare different subset selection strategies discussed in section 3.4.2. It can be observed that for all the methods Mean-of-Features (MoF)

subset selection procedure, introduced in iCaRL [110], performs the best. Surprisingly, *uniform* sampling, despite being simple, is as good as more complex MoF, Plane Distance (PD) and entropy-based sampling strategies. Furthermore, the regularized methods remain insensitive to different sampling strategies, whereas in Vanilla, performance varies a lot against different strategies. We believe this is due to the unconstrained change in the last layer weights of the previous tasks.

3.7 Conclusion

In this chapter, we analyzed the challenges in the continual learning problem, namely, catastrophic forgetting and intransigence, and introduced metrics to quantify them. Such metrics reflect the interplay between *forgetting* and *intransigence*, which we believe will encourage future research for exploiting model capacity, such as, sparsity enforcing regularization, and exploration-based methods for continual learning. In addition, we have presented an efficient version of EWC referred to as EWC++, and a generalization of EWC++ and PI with a KL-divergence-based perspective. Experimentally, we observed that these parameter regularization methods suffer from high intransigence in the *single-head* setting and showed that this can be alleviated with a small subset of representative samples. Since these methods are memory efficient compared to knowledge distillation-based algorithms such as iCaRL, future research in this direction would enable the possibility of continual learning on segmentation tasks.

Appendix

For the sake of completeness, we first give more details on the KL-divergence approximation using Fisher information matrix (section 3.2.3). In particular, we give the proof of KL approximation, $D_{\mathrm{KL}}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta$, discuss the difference between the true Fisher and the empirical Fisher[1], and explain why the Fisher goes to zero at a minimum. Later, in section 3.B.1, we provide a comparison with GEM [87] and show that RWalk significantly outperforms it. Note that, comparison with GEM is not available in the main chapter 3. Additionally, we discuss the sensitivity of different models to the regularization hyperparameter (λ) in section 3.B.2. Finally, we conclude in section 3.B.3 with the details of the architecture and task-based analysis of the network used for CIFAR-100 dataset. We note that with additional experiments and further analysis in this appendix the conclusions of the main chapter 3 hold.

3.A Approximate KL divergence using Fisher Information Matrix

3.A.1 Proof of Approximate KL divergence

Lemma 3.A.1. Assuming $\Delta\theta \to 0$, the second-order Taylor approximation of KL-divergence can be written [7, 102] as:

$$D_{\mathrm{KL}}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta \ , \tag{3.9}$$

where F_θ is the empirical Fisher at θ.

[1]By Fisher, we always mean the empirical Fisher.

Proof. The KL divergence is defined as:

$$D_{\mathrm{KL}}(p_\theta(\mathbf{z})\|p_{\theta+\Delta\theta}(\mathbf{z})) = \mathbb{E}_{\mathbf{z}}\left[\log p_\theta(\mathbf{z}) - \log p_{\theta+\Delta\theta}(\mathbf{z})\right] \ . \tag{3.10}$$

Note that we use the shorthands $p_\theta(\mathbf{z}) = p_\theta(\mathbf{y}|\mathbf{x})$ and $\mathbb{E}_{\mathbf{z}}[\cdot] = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim p_\theta(\mathbf{y}|\mathbf{x})}[\cdot]$. We denote partial derivatives as column vectors. Let us first write the second order Taylor series expansion of $\log p_{\theta+\Delta\theta}(\mathbf{z})$ at θ:

$$\log p_{\theta+\Delta\theta} \approx \log p_\theta + \Delta\theta^\top \frac{\partial \log p_\theta}{\partial \theta} + \frac{1}{2}\Delta\theta^\top \frac{\partial^2 \log p_\theta}{\partial \theta^2}\Delta\theta \ . \tag{3.11}$$

Now, by substituting this in equation 3.10, the KL divergence can be approximated as:

$$\begin{aligned}
D_{\mathrm{KL}}(p_\theta\|p_{\theta+\Delta\theta}) &\approx \mathbb{E}_{\mathbf{z}}[\log p_\theta] - \mathbb{E}_{\mathbf{z}}[\log p_\theta] && \text{(3.12a)}\\
&\quad - \Delta\theta^\top \mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta}{\partial \theta}\right] - \frac{1}{2}\Delta\theta^\top \mathbb{E}_{\mathbf{z}}\left[\frac{\partial^2 \log p_\theta}{\partial \theta^2}\right]\Delta\theta \ , \\
&= \frac{1}{2}\Delta\theta^\top \mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta}{\partial \theta^2}\right]\Delta\theta \quad \text{see equation 3.13} \ , \\
&= \frac{1}{2}\Delta\theta^\top \bar{H}\Delta\theta \quad \text{see equation 3.14b} \ . && \text{(3.12b)}
\end{aligned}$$

In equation 3.12a, since the expectation is taken such that, $\mathbf{x} \sim \mathcal{D}, \mathbf{y} \sim p_\theta(\mathbf{y}|\mathbf{x})$, the first order partial derivatives cancel out, *i.e.*,

$$\begin{aligned}
\mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right] &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\frac{\partial \log p_\theta(\mathbf{y}|\mathbf{x})}{\partial \theta}\right] \ , \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\frac{1}{p_\theta(\mathbf{y}|\mathbf{x})}\frac{\partial p_\theta(\mathbf{y}|\mathbf{x})}{\partial \theta}\right] \ , \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\frac{\partial}{\partial \theta}\sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\right] \ , \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[0] = 0 \ .
\end{aligned} \tag{3.13}$$

Note that this holds for the continuous case as well where, assuming sufficient smoothness and the fact that limits of integration are constants (0 to 1), the Leibniz's rule would allow us to interchange the differentiation and integration operators.

Additionally, in equation 3.12b, the expected value of negative of the Hessian can be shown to be equal to the true Fisher matrix ($\tilde{F}$) by using Information Matrix Equality.

$$\mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta(\mathbf{z})}{\partial \theta^2}\right] = -\mathbb{E}_{\mathbf{z}}\left[\frac{1}{p_\theta(\mathbf{z})}\frac{\partial^2 p_\theta(\mathbf{z})}{\partial \theta^2}\right] \tag{3.14a}$$

$$+ \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^\top\right],$$

$$= -\mathbb{E}_{\mathbf{z}}\left[\frac{1}{p_\theta(\mathbf{z})}\frac{\partial^2 p_\theta(\mathbf{z})}{\partial \theta^2}\right] + \tilde{F}_\theta\ . \tag{3.14b}$$

- By the definition of KL-divergence, the expectation in the above equation is taken such that, $\mathbf{x} \sim \mathcal{D}, \mathbf{y} \sim p_\theta(\mathbf{y}|\mathbf{x})$. This cancels out the first term by following a similar argument as in equation 3.13. Hence, in this case, the expected value of negative of the Hessian equals true Fisher matrix ($\tilde{F}$).

- However, if in equation 3.14b, the expectation is taken such that, $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$, the first term does not go to zero, and $\tilde{F}_\theta$ becomes the *empirical Fisher matrix* (F_θ).

- Additionally, at the optimum, since the model distribution approaches the true data distribution, hence even sampling from dataset *i.e.*, $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$ will make the first term to approach zero, and $\bar{H} \approx F_\theta$.

With the approximation that $\bar{H} \approx \tilde{F}_\theta \approx F_\theta$, the proof is complete.

□

We will argue in section 3.A.2 that the true Fisher matrix is expensive to compute as it requires multiple backward passes. Therefore, as mostly used in literature [7,

102], we also employ empirical Fisher approximation to obtain the KL-divergence.

3.A.2 Empirical vs True Fisher

Loss gradient Let q be any reference distribution and p (parametrized by θ) be the model distribution obtained after applying softmax on the class scores (s). The cross-entropy loss between q and p can be written as: $\ell(\theta) = -\sum_j q_j \log p_j$. The gradients of the loss with respect to the class scores are:

$$\frac{\partial \ell(\theta)}{\partial s_j} = p_j - q_j \ . \tag{3.15}$$

By chain rule, the loss gradients w.r.t. the model parameters are $\frac{\partial \ell(\theta)}{\partial \theta} = \frac{\partial \ell(\theta)}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \theta}$.

Empirical Fisher

In case of an empirical Fisher, the expectation is taken such that $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$. Since every input $\mathbf{x}$ has only one ground truth label, this makes q a Dirac delta distribution . Then, equation 3.15 becomes:

$$\frac{\partial \ell(\theta)}{\partial s_j} = \begin{cases} p_j - 1, & \text{if '}j\text{' is the ground truth label ,} \\ p_j, & \text{otherwise .} \end{cases}$$

Since *at any optimum* the loss-gradient approaches to zero, thus, Fisher being the expected loss-gradient covariance matrix would also approach to a zero matrix.

True Fisher

In case of true Fisher, given $\mathbf{x}$, the expectation is taken such that $\mathbf{y}$ is sampled from the model distributions $p_\theta(\mathbf{y}|\mathbf{x})$. If the final layer of a neural network is a soft-max layer and the network is trained using cross entropy loss, then the output may

be interpreted as a probability distribution over the categorical variables. Thus, at a given θ, the conditional likelihood distribution learned by a neural network is actually a conditional multinoulli distribution defined as $p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{K} p_{\theta,j}^{[y=j]}$, where $p_{\theta,j}$ is the soft-max probability of the j-th class, K are the total number of classes, $\mathbf{y}$ is the one-hot encoding of length K, and $[\cdot]$ is Iverson bracket. At a good optimum, the model distribution $p_\theta(\mathbf{y}|\mathbf{x})$ becomes peaky around the ground truth label, implying $p_{\theta,t} \gg p_{\theta,j}, \forall j \neq t$ where t is the ground-truth label. Thus, given input $\mathbf{x}$, the model distribution $p_\theta(\mathbf{y}|\mathbf{x})$ approach the ground-truth output distribution. This makes the true and empirical Fisher behave in a very similar manner. Note, in order to compute the expectation over the model distribution, the true Fisher requires multiple backward passes making it prohibitively expensive to compute. The standard practice is to resort to the empirical Fisher approximation in this situation [7, 102].

3.B Additional Experiments and Analysis

3.B.1 Comparison with GEM on ResNets

Table 3.2: Following GEM, all the results on ResNets are in the multi-head evaluation setting. Note that, the total number of samples are from all the tasks combined.

Methods	**Total Number of Samples**	$\boldsymbol{A_k}$ (%)
iCaRL	5120	50.8
GEM	5120	65.4
RWalk (Ours)	5000	**70.1**

In this section we show experiments with ResNet18 [53] on CIFAR-100 dataset. In table 3.2 we report the results where we compare our method with iCaRL [110] and Gradient Episodic Memory (GEM) [87]. Both of these methods use ResNet18 as an underlying architecture. Following GEM-setup, we split the CIFAR-100 dataset in 20 tasks where each task consists of 5 consecutive classes, such that

Table 3.3: Comparison of different methods on MNIST and CIFAR-100 as the regularization strength (λ) is varied. With Forgetting and Intransigence we also provide the change (Δ) in the corresponding measures, where the first row in each method is taken as the reference. As discussed in section 3.4.1 in the main chapter 3, RWalk is less sensitive to λ compared to EWC and PI, making it more appealing for continual learning.

Methods	**MNIST**				**CIFAR**			
	λ	A_5(%)	$F_5(\Delta)$	$I_5(\Delta)$	λ	A_{10}(%)	$F_{10}(\Delta)$	$I_{10}(\Delta)$
EWC	75	80.3	0.19 (0)	0.1 (0)	3.0	28.9	0.38 (0)	-0.17 (0)
	75×10^3	79.2	0.15 (-0.04)	0.21 (0.11)	300	34.1	0.28 (-0.1)	-0.07 (0.1)
	75×10^5	79.1	0.13 (-0.06)	0.24 (0.14)	3×10^5	33.7	0.27 (-0.11)	-0.03 (0.14)
PI	0.1	79.3	0.23 (0)	0.05 (0)	0.1	34.7	0.27 (0)	-0.07 (0)
	100	80.3	0.15 (-0.08)	0.22 (0.17)	10	34.3	0.26 (-0.01)	-0.04 (0.03)
	10000	78.5	0.16 (-0.07)	0.18 (0.13)	1×10^4	33.7	0.27 (0)	-0.06 (0.01)
RWalk (Ours)	0.1	82.6	0.16 (0)	0.12 (0)	0.1	34.5	0.28 (0)	-0.06 (0)
	100	81.6	0.16 (0)	0.14 (0.02)	10	33.2	0.28 (0)	-0.06 (0)
	10000	81.6	0.16 (0)	0.12 (0)	1×10^4	34.2	0.28 (0)	-0.05 (0.01)

$\cup_{k=1}^{20} \mathbf{y}^k = \{\{0-4\}, \{5-9\}, \ldots, \{95-99\}\}$. Note, following GEM, all the algorithms are evaluated in multi-head setting (refer section 3.2.1 of the main chapter 3). We refer GEM [87] to report the accuracies of iCaRL and GEM. From the table 3.2, it can be seen that *RWalk outperforms both the methods by a significant margin*.

3.B.2 Effect of regularization hyperparameter

In table 3.3 we analyse the sensitivity of different methods to the regularization hyperparameter (λ). As evident, RWalk is less sensitive to λ compared to EWC [1] [67] and PI [156]. This is because of the normalization of the Fisher and Path-based importance scores in RWalk. For example, as we vary λ by a factor of 1×10^5 on MNIST, the *forgetting* and *intransigence* measures changed by -0.06 and 0.14 on EWC [67], and -0.07 and 0.13 on PI [156], respectively. On the other hand, the change in RWalk, as can be seen in the table 3.3, is 0 for both the measures. On CIFAR-100 a similar trend is observed in table 3.3.

[1] By EWC we always mean its faster version EWC++.

3.B.3 CIFAR Architecture and Task-Level Analysis

In table 3.4 we report the detailed architecture of the convolutional network used in the incremental CIFAR-100 experiments (section 3.6). Note that, in contrast to PI [156], we use only one fully-connected layer (denoted as 'FC' in the table). For each task k, the weights in the last layer of the network is dynamically added. Additionally, in figure 3.6, we present a similar task-level analysis on CIFAR-100 as done for MNIST (figure 3.2 in the main chapter 3). Note that, for all the experiments 'α' in equation 3.6 is set to 0.9 and 'Δt' in equation 3.7 is 10 and 50 for MNIST and CIFAR, respectively.

Table 3.4: CNN architecture for incremental CIFAR-100 used for Vanilla, EWC, PI, iCaRl, RWalk in the main chapter 3. Here, 'n' denotes the number of classes in each task.

Operation	**Kernel**	**Stride**	**Filters**	**Dropout**	**Nonlin.**
3x32x32 input					
Conv	3×3	1×1	32		ReLU
Conv	3×3	1×1	32		ReLU
MaxPool		2×2		0.5	
Conv	3×3	1×1	64		ReLU
Conv	3×3	1×1	64		ReLU
MaxPool		2×2		0.5	
Task 1: FC			n		
$\cdots$: FC			n		
Task k: FC			n		

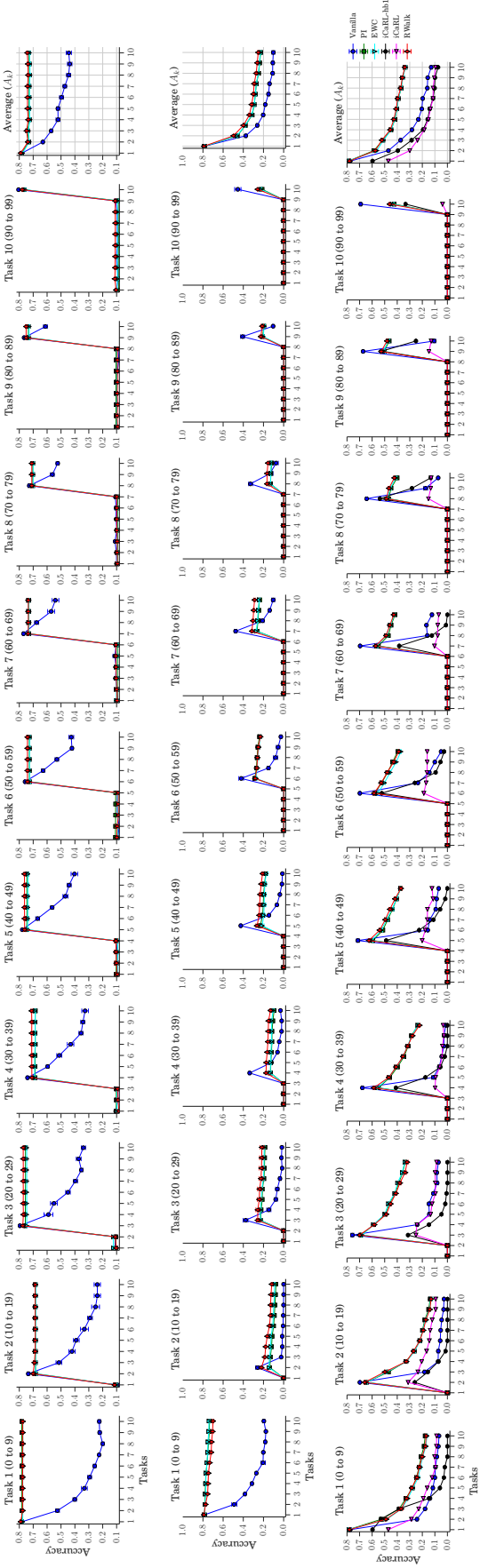


Figure 3.6: Accuracy measure in incremental CIFAR-100 with multi-head evaluation (top), and the single-head evaluation without (middle) and with samples (bottom). The first ten columns show how the performance of different tasks vary as the model is trained for new tasks, *e.g.*, the first plot depicts the variation in performance on Task 1 when the network is sequentially trained for the ten tasks in an incremental manner. The last column shows the average accuracy measure (A_k, by varying k). Mean of features (MoF) sampling is used. (best viewed in color)

Chapter 4

Efficient Continual Learning: Averaged Gradient Episodic Memory

Abstract

In continual learning, the learner is presented with a sequence of tasks, incrementally building a data-driven prior which may be leveraged to speed up learning of a new task. In this work, we investigate the *efficiency* of current continual approaches, in terms of sample complexity, computational and memory cost. Towards this end, we first introduce a new and a more realistic evaluation protocol, whereby learners observe each example only once and hyper-parameter selection is done on a small and disjoint set of tasks, which is not used for the actual learning experience and evaluation. Second, we introduce a new metric measuring how quickly a learner acquires a new skill. Third, we propose an improved version of GEM [87], dubbed Averaged GEM (A-GEM), which enjoys the same or even better performance as GEM, while being almost as computationally and memory efficient as EWC [67] and other regularization-based methods. Finally, we show that all algorithms in-

cluding A-GEM can learn even more quickly if they are provided with task descriptors specifying the classification tasks under consideration. Our experiments on several standard continual learning benchmarks demonstrate that A-GEM has the best trade-off between accuracy and efficiency.

4.1 Introduction

Intelligent systems, whether they are natural or artificial, must be able to quickly adapt to changes in the environment and quickly learn new skills by leveraging past experiences. While current learning algorithms can achieve excellent performance on a variety of tasks, they strongly rely on copious amounts of supervision in the form of labeled data.

The *continual learning* setting attempts at addressing this shortcoming, bringing machine learning closer to a more realistic human learning by acquiring new skills quickly with a small amount of training data, given the experience accumulated in the past. In this setting, the learner is presented with a stream of tasks whose relatedness is not known a priori. The learner has then the potential to learn a new task more quickly, if it can remember how to combine and re-use knowledge acquired while learning related tasks of the past. Of course, for this learning setting to be useful, the model needs to be constrained in terms of amount of compute and memory required. Usually this means that the learner should not be allowed to merely store all examples seen in the past (in which case this reduces the continual learning problem to a multitask problem) nor should the learner be engaged in computations that would not be feasible in real-time, as the goal is to quickly learn from a stream of data.

Unfortunately, the established training and evaluation protocols as well as cur-

rent algorithms for continual learning do not satisfy all the above desiderata, namely learning from a stream of data using limited number of samples, limited memory and limited compute. In the most popular training paradigm, the learner does several passes over the data [67, 3, 118, 126], while ideally the model should need only a handful of samples and these should be provided one-by-one in a single pass [87]. Moreover, when the learner has several hyper-parameters to tune, the current practice is to go over the sequence of tasks several times, each time with a different hyper-parameter value, again ignoring the requirement of learning from a stream of data and, strictly speaking, violating the assumption of the continual learning scenario. While some algorithms may work well in a single-pass setting, they unfortunately require a lot of computation [87] or their memory scales with the number of tasks [118], which greatly impedes their actual deployment in practical applications.

In this chapter, we propose an evaluation methodology and an algorithm that better match our desiderata, namely learning efficiently – in terms of training samples, time and memory – from a stream of tasks. First, we propose a new learning paradigm, whereby the learner performs cross validation on a set of tasks which is disjoint from the set of tasks actually used for evaluation (section 4.2). In this setting, the learner will have to learn and will be tested on an entirely new sequence of tasks and it will perform just a single pass over this data stream. Second, we build upon GEM [87], an algorithm which leverages a small episodic memory to perform well in a single pass setting, and propose a small change to the loss function which makes GEM orders of magnitude faster at training time while maintaining similar performance; we dub this variant of GEM, A-GEM (section 4.4). Third, we explore the use of compositional task descriptors in order to improve the few-shot learning performance within continual learning showing that with this additional information the learner can pick up new skills more quickly (section 4.5).

Fourth, we introduce a new metric to measure the speed of learning, which is useful to quantify the ability of a learning algorithm to learn a new task (section 4.3). And finally, using our new learning paradigm and metric, we demonstrate A-GEM on a variety of benchmarks and against several representative baselines (section 4.6). Our experiments show that A-GEM has a better trade-off between average accuracy and computational/memory cost. Moreover, all algorithms improve their ability to quickly learn a new task when provided with compositional task descriptors, and they do so better and better as they progress through the learning experience.

4.2 Learning Protocol

Currently, most works on continual learning [67, 118, 129, 98] adopt a learning protocol which is directly borrowed from supervised learning. There are T tasks, and each task consists of a training, validation and test sets. During training the learner does as many passes over the data of each task as desired. Moreover, hyper-parameters are tuned on the validation sets by sweeping over the whole sequence of tasks as many times as required by the cross-validation grid search. Finally, metrics of interest are reported on the test set of each task using the model selected by the previous cross-validation procedure.

Since the current protocol violates our stricter definition of continual learning for which the learner can only make a single pass over the data, as we want to emphasize the importance of learning quickly from data, we now introduce a new learning protocol.

We consider two streams of tasks, described by the following ordered sequences of datasets $\mathcal{D}^{CV} = \{\mathcal{D}_1, \cdots, \mathcal{D}_{T^{CV}}\}$ and $\mathcal{D}^{EV} = \{\mathcal{D}_{T^{CV}+1}, \cdots, \mathcal{D}_T\}$, where $\mathcal{D}_k = \{(\mathbf{x}_i^k, t_i^k, y_i^k)_{i=1}^{n_k}\}$ is the dataset of the k-th task, $T^{CV} < T$ (in all our experiments $T^{CV} = 3$ while $T = 20$), and we assume that all datasets are drawn from the same

distribution over tasks. To avoid cluttering of the notation, we let the context specify whether $\mathcal{D}_k$ refers to the training or test set of the k-th dataset.

$\mathcal{D}^{CV}$ is the stream of datasets which will be used during cross-validation; $\mathcal{D}^{CV}$ allows the learner to replay all samples multiple times for the purposes of model hyper-parameter selection. Instead, $\mathcal{D}^{EV}$ is the actual dataset used for final training and evaluation on the test set; the learner will observe training examples from $\mathcal{D}^{EV}$ once and only once, and all metrics will be reported on the test sets of $\mathcal{D}^{EV}$. Since the regularization-based approaches for continual learning [67, 156] are rather sensitive to the choice of the regularization hyper-parameter, we introduced the set $\mathcal{D}^{CV}$, as it seems reasonable in practical applications to have similar tasks that can be used for tuning the system. However, the actual training and testing are then performed on $\mathcal{D}^{EV}$ using a single pass over the data. See algorithm 1 for a summary of the training and evaluation protocol.

Each example in any of these datasets consists of a triplet defined by an input ($\mathbf{x}^k \in \mathcal{X}$), task descriptor ($t^k \in \mathcal{T}$, see section 4.5 for examples) and a target vector ($y^k \in \mathbf{y}^k$), where $\mathbf{y}^k$ is the set of labels specific to task k and $\mathbf{y}^k \subset \mathcal{Y}$. While observing the data, the goal is to learn a predictor $f_\theta : \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{Y}$, parameterized by $\theta \in \mathbb{R}^P$ (a neural network in our case), that can map any test pair $(\mathbf{x}, t)$ to a target y.

4.3 Metrics

Below we describe the metrics used to evaluate the continual learning methods studied in this work. In addition to Average Accuracy (A) and Forgetting Measure (F) [29], we define a new measure, the Learning Curve Area (LCA), that captures how quickly a model learns.

The training dataset of each task, $\mathcal{D}_k$, consists of a total B_k mini-batches. After a presentation of each mini-batch of task k, we evaluate the performance of the

Algorithm 1 Learning and Evaluation Protocols

for h **in** hyper-parameter list **do** ▷ Cross-validation loop, executing multiple passes over $\mathcal{D}^{CV}$
  for $k = 1$ **to** T^{CV} **do** ▷ Learn over data stream $\mathcal{D}^{CV}$ using h
    for $i = 1$ **to** n_k **do** ▷ Single pass over $\mathcal{D}_k$
      Update f_θ using $(\mathbf{x}_i^k, t_i^k, y_i^k)$ and hyper-parameter h
      Update metrics on test set of $\mathcal{D}^{CV}$
    end for
  end for
end for
Select best hyper-parameter setting, h^*, based on average accuracy of test set of $\mathcal{D}^{CV}$, see equation 4.1.
Reset f_θ.
Reset all metrics.
for $k = T^{CV} + 1$ **to** T **do** ▷ Actual learning over datastream $\mathcal{D}^{EV}$
  for $i = 1$ **to** n_k **do** ▷ Single pass over $\mathcal{D}_k$
    Update f_θ using $(\mathbf{x}_i^k, t_i^k, y_i^k)$ and hyper-parameter h^*
    Update metrics on test set of $\mathcal{D}^{EV}$
  end for
end for
Report metrics on test set of $\mathcal{D}^{EV}$.

learner on all the tasks using the corresponding test sets. Let $a_{k,i,j} \in [0, 1]$ be the accuracy evaluated on the test set of task j, after the model has been trained with the i-th mini-batch of task k. Assuming the first learning task in the continuum is indexed by 1 (it will be $T^{CV} + 1$ for $\mathcal{D}^{EV}$) and the last one by T (it will be T^{CV} for $\mathcal{D}^{CV}$), we define the following metrics:

Average Accuracy ($A \in [0, 1]$) Average accuracy after the model has been trained continually with all the mini-batches up till task k is defined as:

$$A_k = \frac{1}{k} \sum_{j=1}^{k} a_{k,B_k,j}. \tag{4.1}$$

In particular, A_T is the average accuracy on all the tasks after the last task has been learned; this is the most commonly used metric used in continual learning.

Forgetting Measure ($F \in [-1, 1]$) [29] Average forgetting after the model has been trained continually with all the mini-batches up till task k is defined as:

$$F_k = \frac{1}{k-1} \sum_{j=1}^{k-1} f_j^k, \tag{4.2}$$

where f_j^k is the forgetting on task 'j' after the model is trained with all the mini-batches up till task k and computed as:

$$f_j^k = \max_{l \in \{1, \cdots, k-1\}} a_{l,B_l,j} - a_{k,B_k,j}. \tag{4.3}$$

Measuring forgetting after all tasks have been learned is important for a two-fold reason. It quantifies the accuracy drop on past tasks, and it gives an indirect notion of how quickly a model may learn a new task, since a forgetful model will have little knowledge left to transfer, particularly so if the new task relates more closely to one of the very first tasks encountered during the learning experience.

Learning Curve Area ($LCA \in [0, 1]$) Let us first define an average b-shot performance (where b is the mini-batch number) after the model has been trained for all the T tasks as:

$$Z_b = \frac{1}{T} \sum_{k=1}^{T} a_{k,b,k}. \tag{4.4}$$

LCA at β is the area of the convergence curve Z_b as a function of $b \in [0, \beta]$:

$$LCA_\beta = \frac{1}{\beta + 1} \int_0^\beta Z_b db = \frac{1}{\beta + 1} \sum_{b=0}^{\beta} Z_b. \tag{4.5}$$

LCA has the following intuitive interpretation. LCA_0 is the average 0-shot performance, the same as forward transfer in Lopez-Paz and Ranzato [87]. LCA_β is the area under the Z_b curve, which is high if the 0-shot performance is good and if the learner learns quickly. In particular, there could be two models with the

same Z_β or A_T, but very different LCA_β because one learns much faster than the other while they both eventually obtain the same final accuracy. This metric aims at discriminating between these two cases, and it makes sense for relatively small values of β since we are interested in models that learn from few examples.

4.4 Averaged Gradient Episodic Memory (A-GEM)

So far we discussed a better training and evaluation protocol for continual learning and a new metric to measure the speed of learning. Next, we review GEM [87], which is an algorithm that has been shown to work well in the single epoch setting. Unfortunately, GEM is very intensive in terms of computational and memory cost, which motivates our efficient variant, dubbed A-GEM. In section 4.5, we will describe how compositional task descriptors can be leveraged to further speed up learning in the few shot regime.

GEM avoids catastrophic forgetting by storing an episodic memory $\mathcal{M}_k$ for each task k. While minimizing the loss on the current task t, GEM treats the losses on the episodic memories of tasks $k < t$, given by $\ell(f_\theta, \mathcal{M}_k) = \frac{1}{|\mathcal{M}_k|} \sum \ell(f_\theta(\mathbf{x}_i, k), y_i)$, as inequality constraints, avoiding their increase but allowing their decrease. This effectively permits GEM to do positive backward transfer which other continual learning methods do not support. Formally, at task t, GEM solves for the following objective:

$$\begin{aligned} \text{minimize}_\theta \quad & \ell(f_\theta, \mathcal{D}_t) \\ \text{s.t.} \quad & \ell(f_\theta, \mathcal{M}_k) \leq \ell(f_\theta^{t-1}, \mathcal{M}_k) \qquad \forall k < t, \end{aligned} \tag{4.6}$$

where f_θ^{t-1} is the network trained till task $t-1$. To inspect the increase in loss, GEM computes the angle between the loss gradient vectors of previous tasks g_k, and the proposed gradient update on the current task g. Whenever the angle is greater

than $90°$ with any of the g_k's, it projects the proposed gradient to the closest in L2 norm gradient $\tilde{g}$ that keeps the angle within the bounds. Formally, the optimization problem GEM solves is given by:

$$\begin{aligned} \text{minimize}_{\tilde{g}} \quad & \frac{1}{2}||g - \tilde{g}||_2^2 \\ \text{s.t.} \quad & \langle \tilde{g}, g_k \rangle \geq 0 \qquad\qquad \forall k < t. \end{aligned} \tag{4.7}$$

Equation 4.7 is a quadratic program (QP) in P-variables (the number of parameters in the network), which for neural networks could be in millions. In order to solve this efficiently, GEM works in the dual space which results in a much smaller QP with only $t - 1$ variables:

$$\begin{aligned} \text{minimize}_{v} \quad & \frac{1}{2} v^\top G G^\top v + g^\top G^\top v \\ \text{s.t.} \quad & v \geq 0, \end{aligned} \tag{4.8}$$

where $G = -(g_1, \cdots, g_{t-1}) \in \mathbb{R}^{(t-1)\times P}$ is computed at each gradient step of training. Once the solution v^* to equation 4.8 is found, the projected gradient update can be computed as $\tilde{g} = G^\top v^* + g$.

While GEM has proven very effective in a single epoch setting [87], the performance gains come at a big computational burden at training time. At each training step, GEM computes the matrix G using all samples from the episodic memory, and solves the QP of equation 4.8. Unfortunately, this inner loop optimization becomes prohibitive when the size of $\mathcal{M}$ and the number of tasks is large, see table 4.7 in Appendix for an empirical analysis. To alleviate the computational burden of GEM, next we propose a much more efficient version of GEM, called Averaged GEM (A-GEM).

Whereas GEM ensures that at every training step the loss of each *individual*

previous task, approximated by the samples in episodic memory, does not increase, A-GEM tries to ensure that at every training step the *average* episodic memory loss over the previous tasks does not increase. Formally, while learning task t, the objective of A-GEM is:

$$\begin{aligned} \text{minimize}_{\theta} \quad & \ell(f_\theta, \mathcal{D}_t) \\ \text{s.t.} \quad & \ell(f_\theta, \mathcal{M}) \leq \ell(f_\theta^{t-1}, \mathcal{M}), \end{aligned} \tag{4.9}$$

where $\mathcal{M} = \cup_{k<t}\mathcal{M}_k$. The corresponding optimization problem reduces to:

$$\begin{aligned} \text{minimize}_{\tilde{g}} \quad & \frac{1}{2}||g - \tilde{g}||_2^2 \\ \text{s.t.} \quad & \tilde{g}^\top g_{ref} \geq 0, \end{aligned} \tag{4.10}$$

where g_{ref} is a gradient computed using a batch randomly sampled from the episodic memory, $(\mathbf{x}_{ref}, y_{ref}) \sim \mathcal{M}$, of all the past tasks. In other words, A-GEM replaces the $t-1$ constraints of GEM with a single constraint, where g_{ref} is the average of the gradients from the previous tasks computed from a random subset of the episodic memory.

The constrained optimization problem of equation 4.10 can now be solved very quickly; when the gradient g violates the constraint, it is projected via:

$$\tilde{g} = g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref}. \tag{4.11}$$

The formal proof of the update rule of A-GEM (equation 4.11) is given in Appendix section 4.B. This makes A-GEM not only memory efficient, as it does not need to store the matrix G, but also orders of magnitude faster than GEM because 1) it is not required to compute the matrix G but just the gradient of a random subset of memory examples, 2) it does not need to solve any QP but just an inner product,

and 3) it will incur in less violations particularly when the number of tasks is large (see table 4.7 and figure 4.6 in Appendix for empirical evidence). All together these factors make A-GEM faster while not hampering its good performance in the single pass setting.

Intuitively, the difference between GEM and A-GEM loss functions is that GEM has better guarantees in terms of worst-case forgetting of each individual task since (at least on the memory examples) it prohibits an increase of any task-specific loss, while A-GEM has better guaratees in terms of average accuracy since GEM may prevent a gradient step because of a task constraint violation although the overall average loss may actually decrease, see Appendix sections 4.D.1 and 4.D.2 for further analysis and empirical evidence. The pseudo-code of A-GEM is given in Appendix algorithm 2.

4.5 Joint Embedding Model Using Compositional Task Descriptors

In this section, we discuss how we can improve forward transfer for all the continual learning methods including A-GEM. In order to speed up learning of a new task, we consider the use of compositional task descriptors where components are shared across tasks and thus allow transfer. Examples of compositional task descriptors are, for instance, a natural language description of the task under consideration or a matrix specifying the attribute values of the objects to be recognized in the task. In our experiments, we use the latter since it is provided with popular benchmark datasets [147, 74]. For instance, if the model has already learned and remembers about two independent properties (e.g., color of feathers and shape of beak), it can quickly recognize a new class provided a descriptor specifying the values of its attributes (yellow feathers and red beak), although this is an entirely unseen

combination.

Borrowing ideas from literature in few-shot learning [75, 157, 37, 152], we learn a joint embedding space between image features and the attribute embeddings. Formally, let $\mathbf{x}^k \in \mathcal{X}$ be the input (e.g., an image), t^k be the task descriptor in the form of a matrix of size $C_k \times A$, where C_k is the number of classes in the k-th task and A is the total number of attributes for each class in the dataset. The joint embedding model consists of a feature extraction module, $\phi_\theta : \mathbf{x}^k \rightarrow \phi_\theta(\mathbf{x}^k)$, where $\phi_\theta(\mathbf{x}^k) \in \mathbb{R}^D$, and a task embedding module, $\psi_\omega : t^k \rightarrow \psi_\omega(t^k)$, where $\psi_\omega(t^k) \in \mathbb{R}^{C_k \times D}$. In this work, $\phi_\theta(.)$ is implemented as a standard multi-layer feed-forward network (see section 4.6 for the exact parameterization), whereas $\psi_\omega(.)$ is implemented as a parameter matrix of dimensions $A \times D$. This matrix can be interpreted as an attribute look-up table as each attribute is associated with a D dimensional vector, from which a class embedding vector is constructed via a linear combination of the attributes present in the class; the task descriptor embedding is then the concatenation of the embedding vectors of the classes present in the task (see Appendix figure 4.9 for the pictorial description of the joint embedding model). During training, the parameters θ and ω are learned by minimizing the cross-entropy loss:

$$\ell_k(\theta, \omega) = \frac{1}{N} \sum_{i=1}^{N} -\log(p(y_i^k | \mathbf{x}_i^k, t^k; \theta, \omega)) \tag{4.12}$$

where $(\mathbf{x}_i^k, t^k, y_i^k)$ is the i-th example of task k. If $y_i^k = c$, then the distribution $p(.)$ is given by:

$$p(c | \mathbf{x}_i^k, t^k; \theta, \omega) = \frac{\exp([\phi_\theta(\mathbf{x}_i^k)\psi_\omega(t^k)^\top]_c)}{\sum_j \exp([\phi_\theta(\mathbf{x}_i^k)\psi_\omega(t^k)^\top]_j)} \tag{4.13}$$

where $[a]_i$ denotes the i-th element of the vector a. Note that the architecture and loss functions are general, and apply not only to A-GEM but also to any other continual learning model (e.g., regularization based approaches). See section 4.6

for the actual choice of parameterization of these functions.

4.6 Experiments

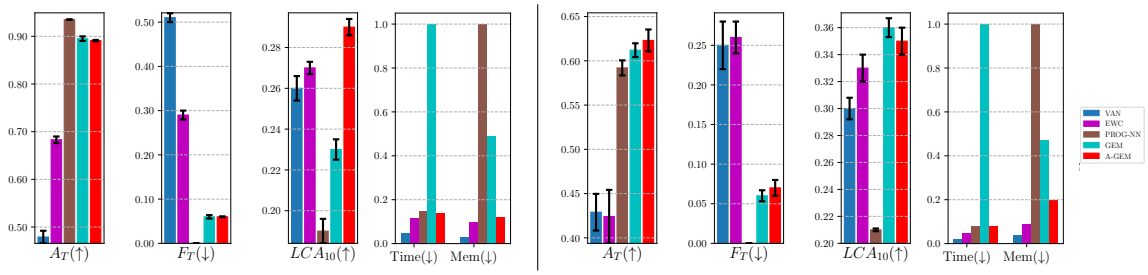
We consider four dataset streams, see table 4.1 in Appendix section 4.A for a summary of the statistics. **Permuted MNIST** [67] is a variant of MNIST [78] dataset of handwritten digits where each task has a certain random permutation of the input pixels which is applied to all the images of that task. **Split CIFAR** [156] consists of splitting the original CIFAR-100 dataset [68] into 20 disjoint subsets, where each subset is constructed by randomly sampling 5 classes *without* replacement from a total of 100 classes. Similarly to Split CIFAR, **Split CUB** is an incremental version of the fine-grained image classification dataset CUB [147] of 200 bird categories split into 20 disjoint subsets of classes. **Split AWA**, on the other hand, is the incremental version of the AWA dataset [74] of 50 animal categories, where each task is constructed by sampling 5 classes *with* replacement from the total 50 classes, constructing 20 tasks. In this setting, classes may overlap among multiple tasks, but within each task they compete against different set of classes. Note that to make sure each training example is only seen once, the training data of each class is split into disjoint sets depending on the frequency of its occurrence in different tasks. For Split AWA, the classifier weights of each class are randomly initialized within each head without any transfer from the previous occurrence of the class in past tasks. Finally, while on Permuted MNIST and Split CIFAR we provide integer task descriptors, on Split CUB and Split AWA we stack together the attributes of the classes (specifying for instance the type of beak, the color of feathers, etc.) belonging to the current task to form a descriptor.

In terms of architectures, we use a fully-connected network with two hidden layers of 256 ReLU units each for Permuted MNIST, a reduced ResNet18 for Split

CIFAR like in Lopez-Paz and Ranzato [87], and a standard ResNet18 [53] for Split CUB and Split AWA. For a given dataset stream, all models use the same architecture, and all models are optimized via stochastic gradient descent with mini-batch size equal to 10. We refer to the joint-embedding model version of these models by appending the suffix '-JE' to the method name.

As described in section 4.2 and outlined in algorithm 1, in order to cross validate we use the first 3 tasks, and then report metrics on the remaining 17 tasks after doing a single training pass over each task in sequence.

Lastly, we compared A-GEM against several baselines and state-of-the-art continual learning approaches which we describe next. **FINETUNE** is a single supervised learning model, trained continually without any regularization, with the parameters of a new task initialized from the parameters of the previous task. **ICARL** [110] is a class-incremental learner that uses nearest-exemplar-based classifier and avoids catastrophic forgetting by regularizing over the feature representation of previous tasks using a knowledge distillation loss. **EWC** [67], **PI** [156], **RWALK** [29] and **MAS** [3] are regularization-based approaches aiming at avoiding catastrophic forgetting by limiting learning of parameters critical to the performance of past tasks. **Progressive Networks** (PROG-NN) [118] is a modular approach whereby a new "column" with lateral connections to previous hidden layers is added once a new task arrives. **GEM** [87] described in section 4.4 is another natural baseline of comparison since A-GEM builds upon it. The amount of episodic memory per task used in ICARL, GEM and A-GEM is set to 250, 65, 50, and 100, and the batch size for the computation of g_{ref} (when the episodic memory is sufficiently filled) in A-GEM is set to 256, 1300, 128 and 128 for MNIST, CIFAR, CUB and AWA, respectively. While populating episodic memory, the samples are chosen uniformly at random for each task. Whereas the network weights are randomly initialized for MNIST, CIFAR and AWA, on the other hand, for CUB, due to



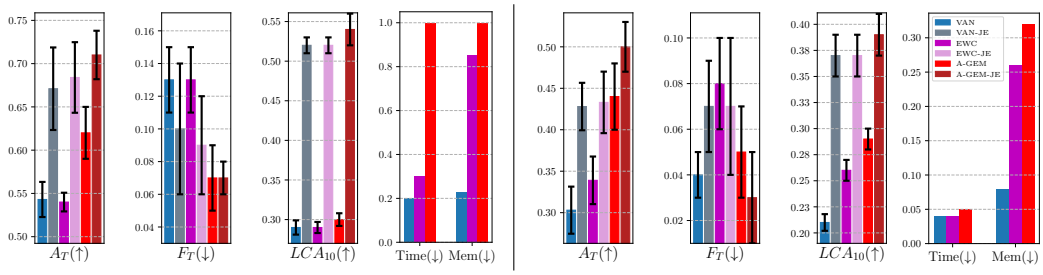
(a) Permuted MNIST (b) Split CIFAR

Figure 4.1: Performance of continual learning models across different measures on Permuted MNIST and Split CIFAR. For Accuracy (A_T) and Learning Curve Measure (LCA_{10}) the higher the number (indicated by ↑) the better is the model. For Forgetting (F_T), Time and Memory the lower the number (indicated by ↓) the better is the model. For Time and Memory, the method with the highest complexity is taken as a reference (value of 1) and the other methods are reported relative to that method. A_T, F_T and LCA_{10} values and confidence intervals are computed over 5 runs. A-GEM provides the best trade-off across different measures and dimensions. Other baselines are given in tables 4.4 and 4.7 in the Appendix, which are used to generate the plots.

the small dataset size, a pre-trained ImageNet model is used. Finally, we consider a multi-task baseline, **MULTI-TASK**, trained on a single pass over shuffled data from all tasks, and thus violating the continual learning assumption. It can be seen as an upper bound performance for average accuracy.

4.6.1 Results

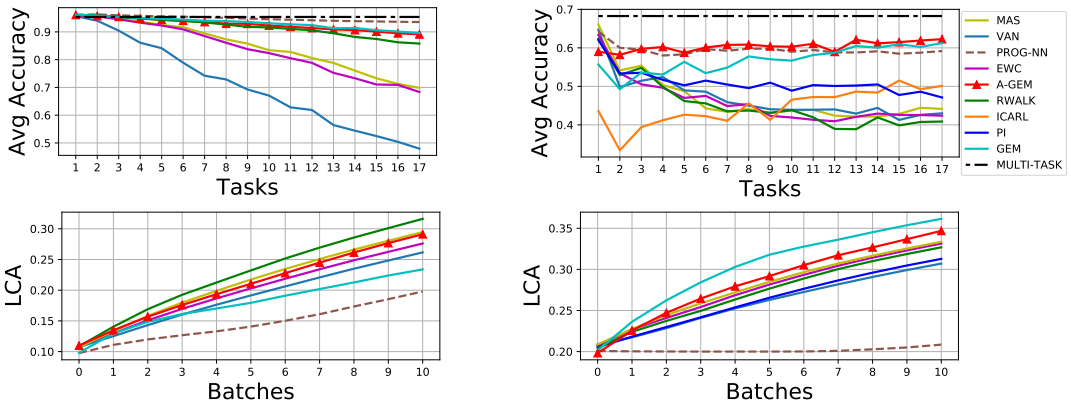
Figures 4.1 and 4.2 show the overall results on all the datasets we considered (for brevity we show only representative methods, see detailed results in Appendix tables 4.4, 4.5, 4.6 and 4.7. First, we observe that A-GEM achieves the best average accuracy on all datasets, except Permuted MNIST, where PROG-NN works better. The reason is because on this dataset each task has a large number of training examples, which enables PROG-NN to learn its task specific parameters and to leverage its lateral connections. However, notice how PROG-NN has the worst memory cost by the end of training - as its number of parameters grows super-linearly with the number of tasks. In particular, in large scale setups (Split CUB



(a) Split CUB (b) Split AWA

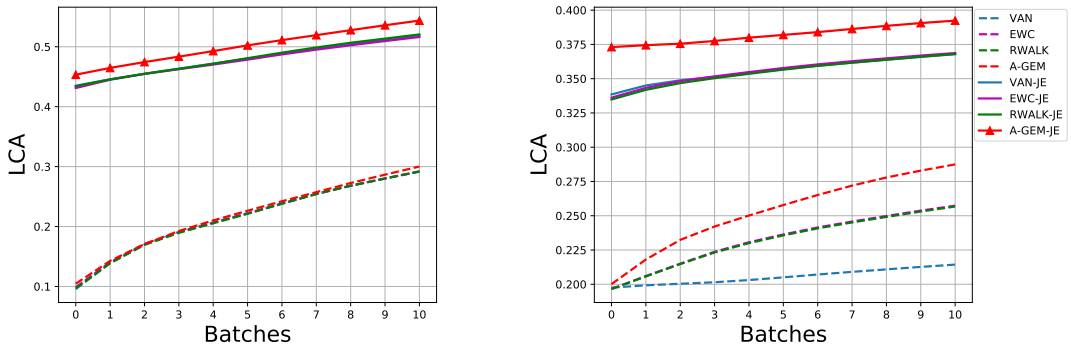
Figure 4.2: Performance of continual learning models across different measures on Split CUB and Split AWA. On both the datasets, PROG-NN runs out of memory. The memory and time complexities of joint embedding models are the same as those of the corresponding standard models and are hence omitted. A_T, F_T and LCA_{10} values and confidence intervals are computed over 10 runs. Other baselines are given in tables 4.5, 4.6, and 4.7 in the Appendix, which are used to generate the plots.

and AWA), PROG-NN runs out of memory during training due to its large size. Also, PROG-NN does not learn well on datasets where tasks have fewer training examples. Second, A-GEM and GEM perform comparably in terms of average accuracy, but A-GEM has much lower time (about 100 times faster) and memory cost (about 10 times lower), comparable to regularization-based approaches like EWC. Third, EWC and similar methods perform only slightly better than FINETUNE on this single pass continual learning setting. The analysis in Appendix section 4.F demonstrates that EWC requires several epochs and over-parameterized architectures in order to work well. Fourth, PROG-NN has no forgetting by construction and A-GEM and GEM have the lowest forgetting among methods that use a fixed capacity architecture. Next, all methods perform similarly in terms of LCA, with PROG-NN being the worst because of its ever growing number of parameters and A-GEM slightly better than all the other approaches. And finally, the use of task descriptors improves average accuracy across the board as shown in figure 4.2, with A-GEM a bit better than all the other methods we tried. All joint-embedding models using task descriptors have better LCA performance, although this is the same across all methods including A-GEM. Overall, we conclude that A-GEM offers



(a) Permuted MNIST (b) Split CIFAR

Figure 4.3: Top Row: Evolution of average accuracy (A_k) as new tasks are learned. Bottom Row: Evolution of LCA during the first ten mini-batches.

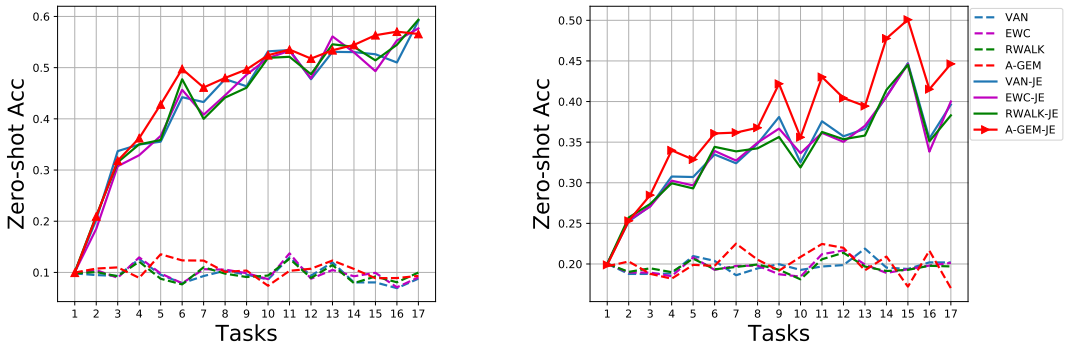


(a) Split CUB (b) Split AWA

Figure 4.4: Evolution of LCA during the first ten mini-batches.

the best trade-off between average accuracy performance and efficiency in terms of sample, memory and computational cost.

Figure 4.3 shows a more fine-grained analysis and comparison with more methods on Permuted MNIST and Split CIFAR. The average accuracy plots show how A-GEM and GEM greatly outperform other approaches, with the exception of PROG-NN on MNIST as discussed above. On different datasets, different methods are best in terms of LCA, although A-GEM is always top-performing. Figure 4.4 shows in more detail the gain brought by task descriptors which greatly speed up learning in the few-shot regime. On these datasets, A-GEM performs the best or on par to the best.



(a) Split CUB (b) Split AWA

Figure 4.5: Evolution of zero-shot performance as the learner sees new tasks on Split CUB and Split AWA datasets.

Finally, in figure 4.5, we report the 0-shot performance of continual learning methods on Split CUB and Split AWA datasets over time, showing a clear advantage of using compositional task descriptors with joint embedding models, which is more significant for A-GEM. Interestingly, the zero-shot learning performance of joint embedding models improves over time, indicating that these models get better at forward transfer or, in other words, become more *efficient* over time.

4.7 Related Work

Continual [112] or Lifelong Learning (LLL) [140] have been the subject of extensive study over the past two decades. One approach to continual learning uses modular compositional models [40, 2, 115, 28, 153, 41], which limit interference among tasks by using different subset of modules for each task. Unfortunately, these methods require searching over the space of architectures which is not sample efficient with current methods. Another approach is to regularize parameters important to solve past tasks [67, 156, 29], which has been proven effective for over-parameterized models in the multiple epoch setting (see Appendix section 4.F), while we focus on learning from few examples using memory efficient models. Methods based on episodic memory [110, 87] require a little bit more memory at training time but can

work much better in the single pass setting we considered [87].

The use of task descriptors for continual learning has already been advocated by Isele et al. [61] but using a sparse coding framework which is not obviously applicable to deep nets in a computationally efficient way, and also by Lopez-Paz and Ranzato [87] although they did not explore the use of compositional descriptors. More generally, tasks descriptors have been used in Reinforcement Learning with similar motivations by several others [136, 121, 16], and it is also a key ingredient in all the zero/few-shot learning algorithms [75, 152, 37, 147, 74].

4.8 Conclusion

In this chapter, we studied the problem of efficient continual Learning in the case where the learner can only do a single pass over the input data stream. We found that our approach, A-GEM, has the best trade-off between average accuracy by the end of the learning experience and computational/memory cost. Compared to the original GEM algorithm, A-GEM is about 100 times faster and has 10 times less memory requirements; compared to regularization based approaches, it achieves significantly higher average accuracy. We also demonstrated that by using compositional task descriptors all methods can improve their few-shot performance, with A-GEM often being the best.

Our detailed experiments reported in Appendix section 4.E also show that there is still a substantial performance gap between continual learning methods, including A-GEM, trained in a sequential learning setting and the same network trained in a non-sequential multi-task setting, despite seeing the same data samples. Moreover, while task descriptors do help in the few-shot learning regime, the LCA performance gap between different methods is very small; suggesting a poor ability of current methods to transfer knowledge even when forgetting has been

eliminated. Addressing these two issues further could be interesting directions for future research.

Appendix

In section 4.A we report the summary of datasets used for the experiments. Section 4.B provides the proof of update rule of A-GEM discussed in section 4.4 of the main chapter 4 and section 4.C details our A-GEM algorithm. In section 4.D, we analyze the differences between A-GEM and GEM, and describe another variation of GEM, dubbed Stochastic GEM (S-GEM). The detailed results of the experiments which were used to generate figures 4.1 and 4.2 in the main chapter 4 are given in section 4.E. In section 4.F, we provide empirical evidence to the conjecture that regularization-based approaches like EWC require over-parameterized architectures and multiple passes over data in order to perform well as discussed in the section 4.6.1 of the main chapter 4. In section 4.G, we provide the grid used for the cross-validation of different hyper-parameters and report the optimal values for different models. Finally, in section 4.H, we pictorially describe the joint embedding model discussed in section 4.5.

4.A Dataset Statistics

Table 4.1: Dataset statistics.

	Perm. MNIST	**Split CIFAR**	**Split CUB**	**Split AWA**
num. of tasks	20	20	20	20
input size	1×28×28	3×32×32	3×224×224	3×224×224
num. of classes per task	10	5	10	5
num. of training images per task	60000	2500	300	-
num. of test images per task	10000	500	290	560

4.B A-GEM Update Rule

Here we provide the proof of the update rule of A-GEM (equation 4.11), $\tilde{g} = g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref}$, stated in section 4.4 of the main chapter 4.

Proof. The optimization objective of A-GEM as described in the equation 4.10 of the main chapter 4, is:

$$\begin{aligned} \text{minimize}_{\tilde{g}} \quad & \frac{1}{2}||g - \tilde{g}||_2^2 \\ \text{s.t.} \quad & \tilde{g}^\top g_{ref} \geq 0. \end{aligned} \tag{4.14}$$

Replacing $\tilde{g}$ with z and rewriting equation 4.14 yields:

$$\begin{aligned} \text{minimize}_{z} \quad & \frac{1}{2}z^\top z - g^\top z \\ \text{s.t.} \quad & -z^\top g_{ref} \leq 0. \end{aligned} \tag{4.15}$$

Note that we discard the term $g^\top g$ from the objective and change the sign of the inequality constraint. The Lagrangian of the constrained optimization problem defined above can be written as:

$$\mathcal{L}(z, \alpha) = \frac{1}{2}z^\top z - g^\top z - \alpha z^\top g_{ref}. \tag{4.16}$$

Now, we pose the dual of equation 4.16 as:

$$\theta_{\mathcal{D}}(\alpha) = \min_z \mathcal{L}(z, \alpha). \tag{4.17}$$

Lets find the value z^* that minimizes the $\mathcal{L}(z, \alpha)$ by setting the derivatives of $\mathcal{L}(z, \alpha)$ w.r.t. to z to zero:

$$\begin{aligned} \nabla_z \mathcal{L}(z, \alpha) &= 0 \\ z^* &= g + \alpha g_{ref}. \end{aligned} \tag{4.18}$$

The simplified dual after putting the value of z^* in equation 4.17 can be written

as:

$$\begin{aligned}\theta_{\mathcal{D}}(\alpha) &= \frac{1}{2}(g^\top g + 2\alpha g^\top g_{ref} + \alpha^2 g_{ref}^\top g_{ref}) - g^\top g - 2\alpha g^\top g_{ref} - \alpha^2 g_{ref}^\top g_{ref} \\ &= -\frac{1}{2}g^\top g - \alpha g^\top g_{ref} - \frac{1}{2}\alpha^2 g_{ref}^\top g_{ref}.\end{aligned}$$

The solution $\alpha^* = \max_{\alpha;\alpha>0} \theta_{\mathcal{D}}(\alpha)$ to the dual is given by:

$$\begin{aligned}\nabla_\alpha \theta_{\mathcal{D}}(\alpha) &= 0 \\ \alpha^* &= -\frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}}.\end{aligned}$$

By putting α^* in equation 4.18, we recover the A-GEM update rule:

$$z^* = g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref} = \tilde{g}.$$

□

4.C A-GEM Algorithm

Algorithm 2 describes the pseudo code for training and evaluation of A-GEM algorithm.

4.D Analysis of GEM and A-GEM

In this section, we empirically analyze the differences between A-GEM and GEM, and report experiments with another computationally efficient but worse performing version of GEM.

Algorithm 2 Training and evaluation of A-GEM on sequential data $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$

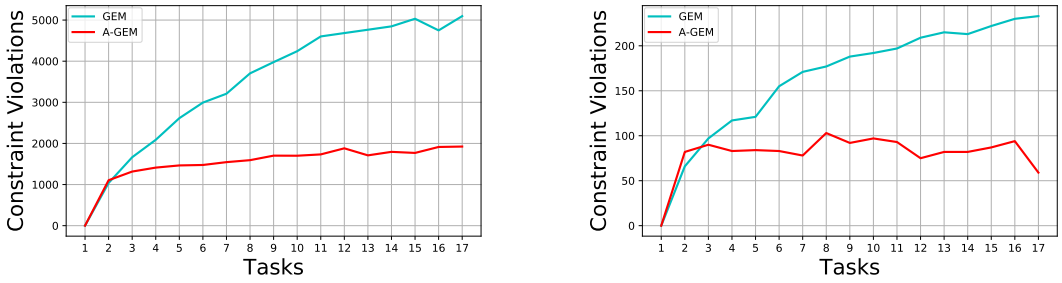
procedure TRAIN($f_\theta, \mathcal{D}^{train}, \mathcal{D}^{test}$)
  $\mathcal{M} \leftarrow \{\}$
  $A \leftarrow 0 \in \mathbb{R}^{T \times T}$
  for $t = \{1, \cdots, T\}$ **do**
    for $(\mathbf{x}, y) \in \mathcal{D}_t^{train}$ **do**
      $(\mathbf{x}_{ref}, y_{ref}) \sim \mathcal{M}$
      $g_{ref} \leftarrow \nabla_\theta \ell(f_\theta(\mathbf{x}_{ref}, t), y_{ref})$
      $g \leftarrow \nabla_\theta \ell(f_\theta(\mathbf{x}, t), y)$
      if $g^\top g_{ref} \geq 0$ **then**
        $\tilde{g} \leftarrow g$
      else
        $\tilde{g} \leftarrow g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref}$
      end if
      $\theta \leftarrow \theta - \alpha \tilde{g}$
    end for
    $\mathcal{M} \leftarrow$ UPDATEEPSMEM($\mathcal{M}, \mathcal{D}_t^{train}, T$)
    $A_{t,:} \leftarrow$ EVAL($f_\theta, \mathcal{D}^{test}$)
  end for
  return f_θ, A
end procedure

procedure EVAL($f_\theta, \mathcal{D}^{test}$)
  $a \leftarrow 0 \in \mathbb{R}^T$
  for $t = \{1, \cdots, T\}$ **do**
    $a_t \leftarrow 0$
    for $(\mathbf{x}, y) \in \mathcal{D}_t^{test}$ **do**
      $a_t \leftarrow a_t +$ ACCURACY($f_\theta(\mathbf{x}, t), y$)
    end for
    $a_t \leftarrow \frac{a_t}{len(\mathcal{D}_t^{test})}$
  end for
  return a
end procedure

procedure UPDATEEPSMEM($\mathcal{M}, \mathcal{D}_t, T$)
  $s \leftarrow \frac{|\mathcal{M}|}{T}$
  for $i = \{1, \cdots, s\}$ **do**
    $(\mathbf{x}, y) \sim \mathcal{D}_t$
    $\mathcal{M} \leftarrow (\mathbf{x}, y)$
  end for
  return $\mathcal{M}$
end procedure

4.D.1 Frequency of Constraint Violations

Figure 4.6 shows the frequency of constraint violations (see equations 4.8 and 4.10) on Permuted MNIST and Split CIFAR datasets. Note that, the number of gradient updates (training steps) per task on MNIST and CIFAR are 5500 and 250, respectively. As the number of tasks increase, GEM violates the optimization constraints at almost each training step, whereas A-GEM plateaus to a much lower value. Therefore, the computational efficiency of A-GEM not only stems from the fact that it avoids solving a QP at each training step (which is much more expensive than a simple inner product) but also from the fewer number of constraint violations. From the figure, we can also infer that as the number of tasks grows the gap between GEM and A-GEM would grow further. Thus, the computational and memory overhead of GEM over A-GEM, see also table 4.7, gets worse as the number of tasks increases.



(a) MNIST (b) CIFAR

Figure 4.6: Number of constraint violations in GEM and A-GEM on Permuted MNIST and Split CIFAR as new tasks are learned.

4.D.2 Average Accuracy and Worst-Case Forgetting

In table 4.2, we empirically demonstrate the different properties induced by the objective functions of GEM and A-GEM. GEM enjoys lower worst-case task forgetting while A-GEM enjoys better overall average accuracy. This is particularly true on the training examples stored in memory, as on the test set the result is confounded by the generalization error.

Table 4.2: Comparison of average accuracy (A_T) and worst-case forgetting (F_{wst}) on the episodic memory ($\mathcal{M}$) and test set ($\mathcal{D}^{EV}$).

Methods	**MNIST**				**CIFAR**			
	$\mathcal{M}$		$\mathcal{D}^{EV}$		$\mathcal{M}$		$\mathcal{D}^{EV}$	
	A_T	F_{wst}	A_T	F_{wst}	A_T	F_{wst}	A_T	F_{wst}
GEM	99.5	0	89.5	0.10	97.1	0.05	61.2	0.14
A-GEM	99.3	0.008	89.1	0.13	72.1	0.15	62.3	0.15

4.D.3 Stochastic GEM (S-GEM)

In this section we report experiments with another variant of GEM, dubbed Stochastic GEM (S-GEM). The main idea in S-GEM is to randomly sample one constraint, at each training step, from the possible $t-1$ constraints of GEM. If that constraint is violated, the gradient is projected only taking into account that constraint. Formally,

the optimization objective of S-GEM is given by:

$$\begin{aligned} \text{minimize}_{\tilde{g}} \quad & \frac{1}{2}||g - \tilde{g}||_2^2 \\ \text{s.t.} \quad & \langle \tilde{g}, g_k \rangle \geq 0 \quad \text{where } k \sim \{1, \cdots, t-1\} \end{aligned} \tag{4.19}$$

In other words, at each training step, S-GEM avoids the increase in loss of one of the previous tasks sampled randomly. In table 4.3 we report the comparison of GEM, S-GEM and A-GEM on Permuted MNIST and Split CIFAR.

Although, S-GEM is closer in spirit to GEM, as it requires randomly sampling one of the GEM constraints to satisfy, compared to A-GEM, which defines the constraint as the average gradient of the previous tasks, it perform slightly worse than GEM, as can be seen from table 4.3.

Table 4.3: Comparison of different variations of GEM on MNIST Permutations and Split CIFAR.

Methods	**Permuted MNIST**		**Split CIFAR**	
	$A_T(\%)$	F_T	$A_T(\%)$	F_T
GEM	89.5	0.06	61.2	0.06
S-GEM	88.2	0.08	56.2	0.12
A-GEM	89.1	0.06	62.3	0.07

4.E Result Tables

In tables 4.4, 4.5, 4.6 and 4.7 we report the detailed results which were used to generate figures 4.1 and 4.2.

Table 4.4: Comparison with different baselines on Permuted MNIST and Split CIFAR. The value of ∞ is assigned to a metric when the model fails to train with the cross-validated values of hyper-parameters found on the subset of the tasks as discussed in section 4.2 of the main chapter 4. The numbers are averaged across 5 runs using a different seed each time. The results from this table are used to generate figure 4.1 in section 4.6.1 of the main chapter 4.

Methods	**Permuted MNIST**			**Split CIFAR**		
	A_T(%)	F_T	LCA_{10}	A_T(%)	F_T	LCA_{10}
FINETUNE	47.9 ($\pm$ 1.32)	0.51 ($\pm$ 0.01)	0.26 ($\pm$ 0.006)	42.9 ($\pm$ 2.07)	0.25 ($\pm$ 0.03)	0.30 ($\pm$ 0.008)
ICARL	-	-	-	50.1	0.11	-
EWC	68.3 ($\pm$ 0.69)	0.29 ($\pm$ 0.01)	0.27 ($\pm$ 0.003)	42.4 ($\pm$ 3.02)	0.26 ($\pm$ 0.02)	0.33 ($\pm$ 0.01)
PI	∞	∞	∞	47.1 ($\pm$ 4.41)	0.17 ($\pm$ 0.04)	0.31 ($\pm$ 0.008)
MAS	69.6 ($\pm$ 0.93)	0.27 ($\pm$ 0.01)	0.29 ($\pm$ 0.003)	44.2 ($\pm$ 2.39)	0.25 ($\pm$ 0.02)	0.33 ($\pm$ 0.009)
RWALK	85.7 ($\pm$ 0.56)	0.08 ($\pm$ 0.01)	**0.31** ($\pm$ 0.005)	40.9 ($\pm$ 3.97)	0.29 ($\pm$ 0.04)	0.32 ($\pm$ 0.005)
PROG-NN	**93.5** ($\pm$ 0.07)	**0**	0.19 ($\pm$ 0.006)	59.2 ($\pm$ 0.85)	**0**	0.21 ($\pm$ 0.001)
GEM	89.5 ($\pm$ 0.48)	0.06 ($\pm$ 0.004)	0.23 ($\pm$ 0.005)	61.2 ($\pm$ 0.78)	0.06 ($\pm$ 0.007)	**0.36** ($\pm$ 0.007)
A-GEM (Ours)	89.1 ($\pm$ 0.14)	0.06 ($\pm$ 0.001)	0.29 ($\pm$ 0.004)	**62.3** ($\pm$ 1.24)	0.07 ($\pm$ 0.01)	0.35 ($\pm$ 0.01)
MULTI-TASK	95.3	-	-	68.3	-	-

Table 4.7: Computational cost and memory complexity of different continual learning approaches. The timing refers to training time on a GPU device. Memory cost is provided in terms of the total number of parameters P, the size of the minibatch B, the total size of the network hidden state H (assuming all methods use the same architecture), the size of the episodic memory M per task. The results from this table are used to generate figures 4.1 and 4.2 in section 4.6.1 of the main chapter 4.

Methods	**Training Time [s]**				**Memory**	
	MNIST	CIFAR	CUB	AWA	Training	Testing
FINETUNE	186	105	54	4123	P + B*H	P + B*H
EWC	403	250	72	4136	4*P + B*H	P + B*H
PROGRESSIVE NETS	510	409	∞	∞	2*P*T + B*H*T	2*P*T + B*H*T
GEM	3442	5238	-	-	P*T + (B+M)*H	P + B*H
A-GEM (Ours)	477	449	420	5221	2*P + (B+M)*H	P + B*H

4.F Analysis of EWC

In this section we provide empirical evidence to the conjecture that regularization-based approaches like EWC need over-parameterized architectures and multiple passes over the samples of each task in order to perform well. The intuition as to why models need to be over-parameterized is because it is easier to avoid cross-task

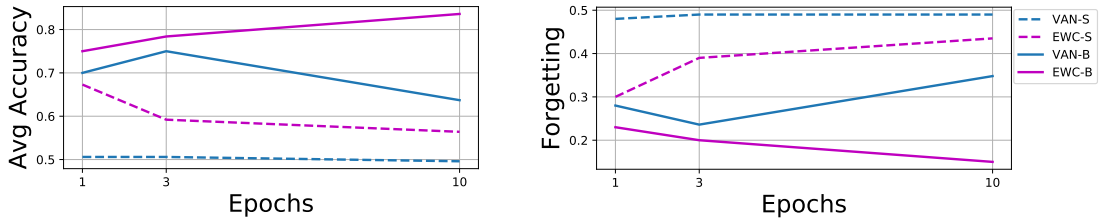
Table 4.5: Average accuracy and forgetting of standard models (left) and joint embedding models (right) on Split CUB. The value of 'OoM' is assigned to a metric when the model fails to fit in the memory. The numbers are averaged across 10 runs using a different seed each time. The results from this table are used to generate figure 4.2 in section 4.6.1 of the main chapter 4.

Methods	**Split CUB**		
	A_T(%)	F_T	LCA_{10}
FINETUNE	54.3 (± 2.03) / 67.1 (± 4.77)	0.13 (± 0.02)/ 0.10 (± 0.04)	0.29 (± 0.009) / 0.52 (± 0.01)
EWC	54 (± 1.08) / 68.4 (± 4.08)	0.13 (± 0.02) / 0.09 (± 0.03)	0.29 (± 0.007) / 0.52 (± 0.01)
PI	55.3 (± 2.28) / 66.6 (± 5.18)	0.12 (± 0.02)/ 0.10 (± 0.04)	0.29 (± 0.008) / 0.52 (± 0.01)
RWALK	54.4 (± 1.82) / 67.4 (± 3.50)	0.13 (± 0.01) / 0.10 (± 0.03)	0.29 (± 0.008) / 0.52 (± 0.01)
PROG-NN	OoM / OoM	OoM / OoM	OoM / OoM
A-GEM (Ours)	**62** (± 3.5) / **71** (± 2.83)	**0.07** (± 0.02) / **0.07** (± 0.01)	**0.30** (± 0.008) / **0.54** (± 0.02)
MULTI-TASK	65.6 / 73.8	- / -	- / -

Table 4.6: Average accuracy and forgetting of standard models (left) and joint embedding models (right) on Split AWA. The value of 'OoM' is assigned to a metric when the model fails to fit in the memory. The numbers are averaged across 10 runs using a different seed each time. The results from this table are used to generate figure 4.2 in section 4.6.1 of the main chapter 4.

Methods	**Split AWA**		
	A_T(%)	F_T	LCA_{10}
FINETUNE	30.3 (± 2.84) / 42.8 (± 2.86)	**0.04** (± 0.01) / 0.07 (± 0.02)	0.21 (± 0.008) / 0.37 (± 0.02)
EWC	33.9 (± 2.87) / 43.3 (± 3.71)	0.08 (± 0.02) / 0.07 (± 0.03)	0.26 (± 0.01) / 0.37 (± 0.02)
PI	33.9 (± 3.25) / 43.4 (± 3.49)	0.08 (± 0.02) / 0.06 (± 0.02)	0.26 (± 0.01) / 0.37 (± 0.02)
RWALK	33.9 (± 2.91) / 42.9 (± 3.10)	0.08 (± 0.02) / 0.07 (± 0.02)	0.26 (± 0.01) / 0.37 (± 0.02)
PROG-NN	OoM / OoM	OoM / OoM	OoM / OoM
A-GEM (Ours)	**44** (± 4.10) / **50** (± 3.25)	0.05 (± 0.02) / **0.03** (± 0.02)	**0.29** (± 0.01) / **0.39** (± 0.02)
MULTI-TASK	64.8 / 66.8	- / -	- / -

interference when the model has additional capacity. In the single-pass setting and when each task does not have very many training samples, regularization-based appraches also suffer because regularization parameters cannot be estimated well from a model that has not fully converged. Moreover, for tasks that do not have much data, regularization-based approaches do not enable any kind of positive backward transfer [87] which further hurts performance as the predictor cannot leverage knowledge acquired later to improve its prediction on past tasks. Finally, regularization-based approaches perform much better in the multi-epoch setting simply because in this setting the baseline un-regularized model performs much



(a) Accuracy (b) Forgetting

Figure 4.7: Permuted MNIST: Change in average accuracy and forgetting as the number of epochs are increased. Tokens '-S' and '-B' denote smaller and bigger networks, respectively.

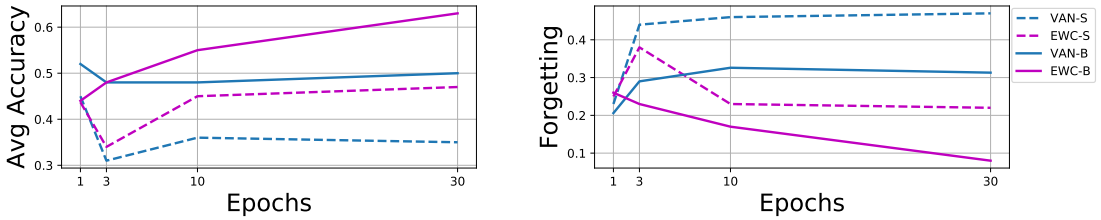
worse, as it overfits much more to the data of the current task, every time unlearning what it learned before.

We consider Permuted MNIST and Split CIFAR datasets as described in section 4.6 of the main chapter 4. For MNIST, the two architecture variants that we experiment with are; 1) two-layer fully-connected network with 256 units in each layer (denoted by $-S$ suffix), and 2) two-layer fully-connected network with 2000 units in each layer (denoted by $-B$ suffix).

For CIFAR, the two architecture variants are; 1) ResNet-18 with 3 times less feature maps in all the layers (denoted by $-S$ suffix), and 2) Standard ResNet-18 (denoted by $-B$ token).

We run the experiments on FINETUNE and EWC with increasing the number of epochs from 1 to 10 for Permuted MNIST and from 1 to 30 for CIFAR. For instance, when epoch is set to 10, it means that the training samples of task t are presented 10 times before showing examples from task $t+1$. In figures 4.7 and 4.8 we plot the Average Accuracy (equation 4.1) and Forgetting (equation 4.2) on Permuted MNIST and Split CIFAR, respectively.

We observe that the average accuracy significantly improves with the number of epochs only when EWC is applied to the big network. In particular, in the single epoch setting, EWC peforms similarly to the baseline FINETUNE on Split CIFAR which has fewer number of training examples per task.



(a) Accuracy (b) Forgetting

Figure 4.8: Split CIFAR: Change in average accuracy and forgetting as the number of epochs are increased. Tokens '-S' and '-B' denote smaller and bigger networks, respectively.

4.G Hyper-parameter Selection

Below we report the hyper-parameters grid considered for different experiments. Note, as described in the section 4.6 of the main chapter 4, to satisfy the requirement that a learner does not see the data of a task more than once, first T^{CV} tasks are used to cross-validate the hyper-parameters. In all the datasets, the value of T^{CV} is set to '3'. The best setting for each experiment is reported in the parenthesis.

- MULTI-TASK
  - learning rate: [0.3, 0.1, 0.03 (MNIST perm, Split CIFAR, Split CUB, Split AWA), 0.01, 0.003, 0.001, 0.0003, 0.0001]
- MULTI-TASK-JE
  - learning rate: [0.3, 0.1, 0.03 (Split CUB, Split AWA), 0.01, 0.003, 0.001, 0.0003, 0.0001]
- FINETUNE
  - learning rate: [0.3, 0.1, 0.03 (MNIST perm, Split CUB), 0.01 (Split CIFAR), 0.003, 0.001 (Split AWA), 0.0003, 0.0001]
- VAN-JE

  - learning rate: [0.3, 0.1, 0.03 (Split CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]

- PROG-NN

  - learning rate: [0.3, 0.1 (MNIST perm,), 0.03 (Split CIFAR, Split AWA), 0.01 (Split CUB), 0.003, 0.001, 0.0003, 0.0001]

- EWC

  - learning rate: [0.3, 0.1, 0.03 (MNIST perm, Split CIFAR, Split CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [1 (Split CUB), 10 (MNIST perm, Split CIFAR), 100 (Split AWA), 1000, 10000]

- EWC-JE

  - learning rate: [0.3, 0.1, 0.03 (Split CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [1, 10 (Split CUB), 100 (Split AWA), 1000, 10000]

- PI

  - learning rate: [0.3, 0.1 (MNIST perm), 0.03 (Split CUB), 0.01 (Split CIFAR), 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [0.001, 0.01, 0.1 (MNIST perm, Split CIFAR, Split CUB), 1 (Split AWA), 10]

- PI-JE

  - learning rate: [0.3, 0.1, 0.03 (Split CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [0.001, 0.01, 0.1 (Split CUB), 1, 10 (Split AWA)]

- MAS
  - learning rate: [0.3, 0.1 (MNIST perm), 0.03 (Split CIFAR, Split CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [0.01, 0.1 (MNIST perm, Split CIFAR, Split CUB), 1 (Split AWA), 10]
- MAS-JE
  - learning rate: [0.3, 0.1, 0.03 (Split CUB), 0.01, 0.003, 0.001 (Split AWA), 0.0003, 0.0001]
  - regularization: [0.01, 0.1 (Split CUB, Split AWA), 1, 10]
- RWALK
  - learning rate: [0.3, 0.1 (MNIST perm), 0.03 (Split CIFAR, Split CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [0.1, 1 (MNIST perm, Split CIFAR, Split CUB), 10 (Split AWA), 100, 1000]
- RWALK-JE
  - learning rate: [0.3, 0.1, 0.03 (SPLIT CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]
  - regularization: [0.1, 1 (Split CUB), 10 (Split AWA), 100, 1000]
- A-GEM
  - learning rate: [0.3, 0.1 (MNIST perm), 0.03 (Split CIFAR, Split CUB), 0.01 (Split AWA), 0.003, 0.001, 0.0003, 0.0001]
- A-GEM-JE

- learning rate: [0.3, 0.1, 0.03 (SPLIT CUB), 0.01, 0.003 (Split AWA), 0.001, 0.0003, 0.0001]

4.H Pictorial Description of Joint Embedding Model

In figure 4.9 we provide a pictorial description of the joint embedding model discussed in the section 4.5 of the main chapter 4.

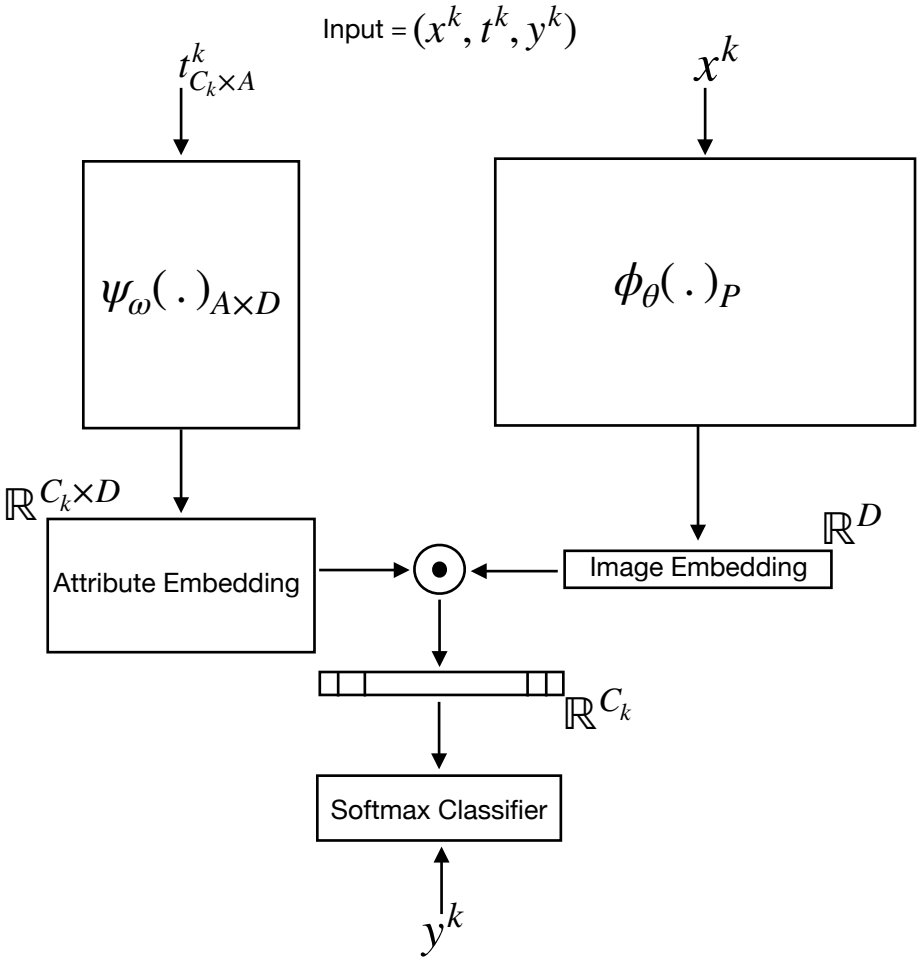


Figure 4.9: Pictorial description of the joint embedding model discussed in the section 4.5 of the main chapter 4. Modules; $\phi_\theta(.)$ and $\psi_\omega(.)$ are implemented as feed-forward neural networks with P and $A \times D$ parameters, respectively. The descriptor of task k (t^k) is a matrix of dimensions $C_k \times A$, shared among all the examples of the task, constructed by concatenating the A-dimensional class attribute vectors of C_k classes in the task.

Chapter 5

Continual Learning by Directly Replaying the Episodic Memory

Abstract

In continual learning (CL), an agent learns from a stream of tasks leveraging prior experience to transfer knowledge to future tasks. It is an ideal framework to decrease the amount of supervision in the existing learning algorithms. But for a successful knowledge transfer, the learner needs to remember how to perform previous tasks. One way to endow the learner the ability to perform tasks seen in the past is to store a small memory, dubbed episodic memory, that stores few examples from previous tasks and then to replay these examples when training for future tasks. In this chapter, we empirically analyze the effectiveness of a very small episodic memory in a CL setup where each training example is only seen once. Surprisingly, across four rather different supervised learning benchmarks adapted to CL, a very simple baseline, that jointly trains on both examples from the current task as well as examples stored in the episodic memory, significantly outperforms specifically designed

CL approaches with and without episodic memory. Interestingly, we find that repetitive training on even tiny memories of past tasks does not harm generalization, on the contrary, it improves it, with gains between 7% and 17% when the memory is populated with a single example per class.

5.1 Introduction

The objective of continual learning (CL) is to rapidly learn new skills from a sequence of tasks leveraging the knowledge accumulated in the past. Catastrophic forgetting [93], i.e. the inability of a model to recall how to perform tasks seen in the past, makes such efficient adaptation extremely difficult.

This decades old problem of CL [112, 140] is now seeing a surge of interest in the research community with several methods proposed to tackle catastrophic forgetting [110, 67, 156, 80, 3, 87, 82, 30]. In this chapter, we quantitatively study some of these methods (that assume a fixed network architecture) on four benchmark datasets under the following assumptions: i) each task is fully supervised, ii) each example from a task can only be seen once using the learning protocol proposed by Chaudhry et al. [30] (see section 5.3), and iii) the model has access to a small memory storing examples of past tasks. Restricting the size of such episodic memory is important because it makes the continual learning problem more realistic and distinct from multi-task learning where complete datasets of all the tasks are available at each step.

We empirically observe that a very simple baseline, dubbed Experience Replay (ER)[1], that jointly trains on both the examples from the current task and examples stored in the very small episodic memory not only gives superior performance

[1]For consistency to prior work in the literature, we will refer to this approach which trains on the episodic memory as ER, although its usage for supervised learning tasks is far less established.

over the existing state-of-the-art approaches specifically designed for CL (with and without episodic memory), but it also is computationally very efficient. We verify this finding on four rather different supervised learning benchmarks adapted for CL; Permuted MNIST, Split CIFAR, Split miniImageNet and Split CUB. Importantly, repetitive training on the same examples of a tiny episodic memory does not harm generalization on past tasks. In section 5.5.5, we analyze this phenomenon and provide insights as to why directly training on the episodic memory does not have a detrimental effect in terms of generalization. Briefly, we observe that the training on the datasets of subsequent tasks acts like a data-dependent regularizer on past tasks allowing the repetitive training on tiny memory to generalize beyond the episodic memory. We further observe that methods, that do not train directly on the memory, such as GEM [87] and A-GEM [30], underfit the training data and end up not fully utilizing the beneficial effects of this implicit and data depdendent regularization.

Overall, ER with tiny episodic memories offers very strong performance at a very small additional computational cost over the fine-tuning baseline. We believe that this approach will serve as a stronger baseline for the development of future CL approaches.

5.2 Related Work

Regularization-based CL approaches These works attempt to reduce forgetting by regularizing the objective such that it either penalizes the feature drift on already learned tasks [85, 110] or discourages change in parameters that were important to solve past tasks [67, 156, 29, 3]. The former approach relies on the storage of network activations and subsequent deployment of knowledge distillation [57], whereas the latter approach stores a measure of parameter importance whose best

case memory complexity is the same as the total number of network parameters.

Memory-Based CL approaches These approaches [87, 111, 30] use episodic memory that stores a subset of data from past tasks to tackle forgetting. One approach to leverage such episodic memory is to use it to constrain the optimization such that the loss on past tasks can never increase [87, 30].

Experience Replay (ER) The use of ER is well established in reinforcement learning (RL) tasks [94, 95, 43, 114]. Isele and Cosgun [60], for instance, explore different ways to populate a relatively large episodic memory for a continual RL setting where the learner does multiple passes over the data. In this work instead, we study supervised learning tasks with a single pass through data and a very small episodic memory. More recently, [52, 111] used ER for supervised CL learning tasks. Hayes et al. [52], independently, study different replay strategies in ER and show improvements over the finetune baseline. Our contribution is to show the improvements brought by ER, perhaps surprisingly, over the specifically designed CL approaches. We differ from [111] in considering episodic memories of much smaller sizes. Finally, and most importantly, we extend these previous studies by analyzing *why* repetitive training on tiny memories does not lead to overfitting (section 5.5.5).

5.3 Learning Framework

5.3.1 Protocol for Single-Pass Through the Data

We use the learning protocol proposed by Chaudhry et al. [30]. There are two streams of tasks, described by the following ordered sequences of datasets, one stream for Cross-*V*alidation $\mathcal{D}^{CV} = \{\mathcal{D}_{-T^{CV}}, \cdots, \mathcal{D}_{-1}\}$ consisting of T^{CV} tasks,

and one for *EV*aluation $\mathcal{D}^{EV} = \{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$ consisting of T tasks, where $\mathcal{D}_k = \{(\mathbf{x}_i^k, t_i^k, y_i^k)_{i=1}^{n_k}\}$ is the dataset of the k-th task. The sequence $\mathcal{D}^{CV}$ contains only a handful of tasks and it is only used for cross-validation purposes. Tasks from this sequence can be replayed as many times as needed and have various degree of similarity to tasks in the training and evaluation dataset, $\mathcal{D}^{EV}$. The latter stream, $\mathcal{D}^{EV}$, instead can be played only once; the learner will observe examples in *sequence* and will be tested throughout the learning experience. The final performance will be reported on the held-out test set drawn from $\mathcal{D}^{EV}$.

The k-th task in any of these streams consists of $\mathcal{D}_k = \{(\mathbf{x}_i^k, t_i^k, y_i^k)_{i=1}^{n_k}\}$, where each triplet constitutes an example defined by an input ($\mathbf{x}^k \in \mathcal{X}$), a task descriptor ($t^k \in \mathcal{T}$) which is an integer id in this work, and a target vector ($y^k \in \mathbf{y}^k$), where $\mathbf{y}^k$ is the set of labels specific to task k and $\mathbf{y}^k \subset \mathcal{Y}$.

5.3.2 Metrics

We measure performance on $\mathcal{D}^{EV}$ using two metrics, as standard practice in the literature [87, 29]:

Average Accuracy ($A \in [0, 1]$) Let $a_{i,j}$ be the performance of the model on the held-out test set of task 'j' after the model is trained on task 'i'. The average accuracy at task T is then defined as:

$$A_T = \frac{1}{T} \sum_{j=1}^{T} a_{T,j} \tag{5.1}$$

Forgetting ($F \in [-1, 1]$) Let f_j^i be the forgetting on task 'j' after the model is trained on task 'i' which is computed as:

$$f_j^i = \max_{l \in \{1, \cdots, i-1\}} a_{l,j} - a_{i,j} \tag{5.2}$$

The average forgetting measure at task T is then defined as:

$$F_T = \frac{1}{T-1} \sum_{j=1}^{T-1} f_j^T \tag{5.3}$$

5.4 Experience Replay

Recent works [87, 30] have shown that methods relying on episodic memory have superior performance than regularization based approaches (e.g., [67, 156]) when using a "single-pass through the data" protocol (section 5.3.1). While GEM [87] and its more efficient version A-GEM [30] used the episodic memory as a mean to project gradients, here we drastically simplify the optimization problem and, similar to Riemer et al. [111] and Hayes et al. [52], directly train on the examples stored in a very small memory, resulting in better performance and more efficient learning.

The overall training procedure is given in algorithm 3. Compared to the simplest baseline model that merely fine-tunes the parameters on the new task starting from the previous task parameter vector, ER makes two modifications. First, it has an episodic memory which is updated at every time step, `line 8`. Second, it doubles the size of the minibatch used to compute the gradient descent parameter update by stacking the actual minibatch of examples from the current task with a minibatch of examples taken at random from the memory, `line 7`. As we shall see in our empirical validation, these two simple modifications yield much better generalization and substantially limit forgetting, while incurring in a negligible additional computational cost on modern GPU devices. Next, we explain the difference between the direct (ER) and indirect (A-GEM) training on episodic memory from the optimization perspective.

Algorithm 3 Experience Replay for Continual Learning.

1: **procedure** ER($\mathcal{D}$, mem_sz, batch_sz, lr)
2: $\mathcal{M} \leftarrow \{\} * \text{mem\_sz}$ ▷ Allocate memory buffer of size mem_sz
3: $n \leftarrow 0$ ▷ Number of training examples seen in the continuum
4: **for** $t \in \{1, \cdots, T\}$ **do**
5: **for** $B_n \overset{K}{\sim} \mathcal{D}_t$ **do** ▷ Sample without replacement a mini-batch of size K from task t
6: $B_{\mathcal{M}} \overset{K}{\sim} \mathcal{M}$ ▷ Sample a mini-batch from $\mathcal{M}$
7: $\theta \leftarrow SGD(B_n \cup B_{\mathcal{M}}, \theta, \text{lr})$ ▷ Single gradient step to update the parameters by stacking current minibatch with minibatch from memory
8: $\mathcal{M} \leftarrow \texttt{UpdateMemory}(\text{mem\_sz}, t, n, B_n)$ ▷ Memory update, see section 5.4
9: $n \leftarrow n + \text{batch\_sz}$ ▷ Counter update
10: **end for**
11: **end for**
12: **return** $\theta, \mathcal{M}$
13: **end procedure**

A-GEM vs ER: Let us assume that B_n is a mini-batch of size K from the current task t and $B_{\mathcal{M}}$ is the same size mini-batch from a very small episodic memory $\mathcal{M}$. Furthermore, following the notation from [30], let g be the gradient computed with mini-batch B_n and g_{ref} be the gradient computed with $B_{\mathcal{M}}$. In A-GEM, if $g^T g_{ref} \geq 0$, then the current task gradient g is directly used for optimization whereas if $g^T g_{ref} < 0$, g is projected such that $g^T g_{ref} = 0$. Refer to Eq. 11 in [30] for the exact form of projection. In ER instead, since both mini-batches are used in the optimization step, the *average* of g and g_{ref} is used. It may seem a bit counter-intuitive that, even though ER repetitively trains on $\mathcal{M}$, it is still able to generalize to previous tasks beyond the episodic memory. We investigate this question in section 5.5.5.

Since we study the usage of tiny episodic memories, the sample that the learner selects to populate the memory becomes crucial, see `line 8` of the algorithm. For this, we describe various strategies to write into the memory. All these strategies assume access to a *continuous* stream of data and a small episodic memory, which

rules out approaches relying on the temporary storage of all the examples seen so far. This restriction is consistent with our definition of CL: a learning experience through a stream of data under the constraint of a fixed and small sized memory and limited compute budget.

Reservoir Sampling: Similarly to Riemer et al. [111], Reservoir sampling [146] takes as input a stream of data of unknown length and returns a random subset of items from that stream. If 'n' is the number of points observed so far and 'mem_sz' is the size of the reservoir (sampling buffer), this selection strategy samples each data point with a probability $\frac{\text{mem\_sz}}{n}$. The routine to update the memory is given in Appendix algorithm 4.

Ring Buffer: Similarly to Lopez-Paz and Ranzato [87], for each task, the ring buffer strategy allocates as many equally sized FIFO buffers as there are classes. If C is the total number of classes across all tasks, and mem_sz is the total size of episodic memory, each stack has a buffer of size $\frac{\text{mem\_sz}}{C}$. As shown in Appendix algorithm 5, the memory stores the last few observations from each class. Unlike reservoir sampling, samples from older tasks do not change throughout training, leading to potentially stronger overfitting. Also, at early stages of training the memory is not fully utilized since each stack has a constant size throughout training. However, this simple sampling strategy guarantees equal representation of all classes in the memory, which is particularly important when the memory is tiny.

k-Means: For each class, we use online k-Means to estimate the k centroids in feature space, using the representation before the last classification layer. We then store in the memory the input examples whose feature representation is the closest to such centroids, see Appendix algorithm 6. This memory writing strategy has similar benefits and drawbacks of ring buffer, except that it has potentially better

coverage of the feature space in L2 sense.

Mean of Features (MoF): Similarly to Rebuffi et al. [110], for each class we compute a running estimate of the average feature vector just before the classification layer and store examples whose feature representation is closest to the average feature vector (see details in Appendix algorithm 7). This writing strategy has the same balancing guarantees of ring buffer and k-means, but it populates the memory differently. Instead of populating the memory at random or using k-Means, it puts examples that are closest to the mode in feature space.

5.5 Experiments

In this section, we review the benchmark datasets used in our evaluation, as well as the architectures and the baselines we compared against. We then report the results we obtained using episodic memory and experience replay (ER). Finally, we conclude with a brief analysis investigating generalization when using ER on tiny memories.

5.5.1 Datasets

We consider four commonly used benchmarks in CL literature. **Permuted MNIST** [67] is a variant of MNIST [78] dataset of handwritten digits where each task has a certain random permutation of the input pixels which is applied to all the images of that task. Our Permuted MNIST benchmark consists of a total of 23 tasks.

Split CIFAR [156] consists of splitting the original CIFAR-100 dataset [68] into 20 disjoint subsets, each of which is considered as a separate task. Each task has 5 classes that are randomly sampled *without* replacement from the total of 100 classes.

Similarly to Split CIFAR, **Split miniImageNet** is constructed by splitting mini-

ImageNet [145], a subset of ImageNet with a total of 100 classes and 600 images per class, to 20 disjoint subsets.

Finally, **Split CUB** [30] is an incremental version of the fine-grained image classification dataset CUB [147] of 200 bird categories split into 20 disjoint subsets of classes.

In all cases, $\mathcal{D}^{CV}$ consists of 3 tasks while $\mathcal{D}^{EV}$ contains the remaining tasks. As described in section 5.3.2, we report metrics on $\mathcal{D}^{EV}$ after doing a single training pass over each task in the sequence. The hyper-parameters selected via cross-validation on $\mathcal{D}^{CV}$ are reported in Appendix table 5.8.

5.5.2 Architectures

For MNIST, we use a fully-connected network with two hidden layers of 256 ReLU units each. For CIFAR and miniImageNet, a reduced ResNet18, similar to Lopez-Paz and Ranzato [87], is used and a standard ResNet18 with ImageNet pretraining is used for CUB. The input integer task id is used to select a task specific classifier head, and the network is trained via cross-entropy loss.

For a given dataset stream, all baselines use the same architecture, and all baselines are optimized via stochastic gradient descent with a mini-batch size equal to 10. The size of the mini-batch sampled from the episodic memory is also set to 10 irrespective of the size of the episodic buffer.

5.5.3 Baselines

We compare against the following baselines:

- FINETUNE, a model trained continually without any regularization and episodic memory, with parameters of a new task initialized from the parameters of the previous task.

- EWC [67], a regularization-based approach that avoids catastrophic forgetting by limiting the learning of parameters critical to the performance of past tasks, as measured by the Fisher information matrix (FIM). In particular, we compute the FIM as a moving average similar to EWC++ in [29] and online EWC in [126].

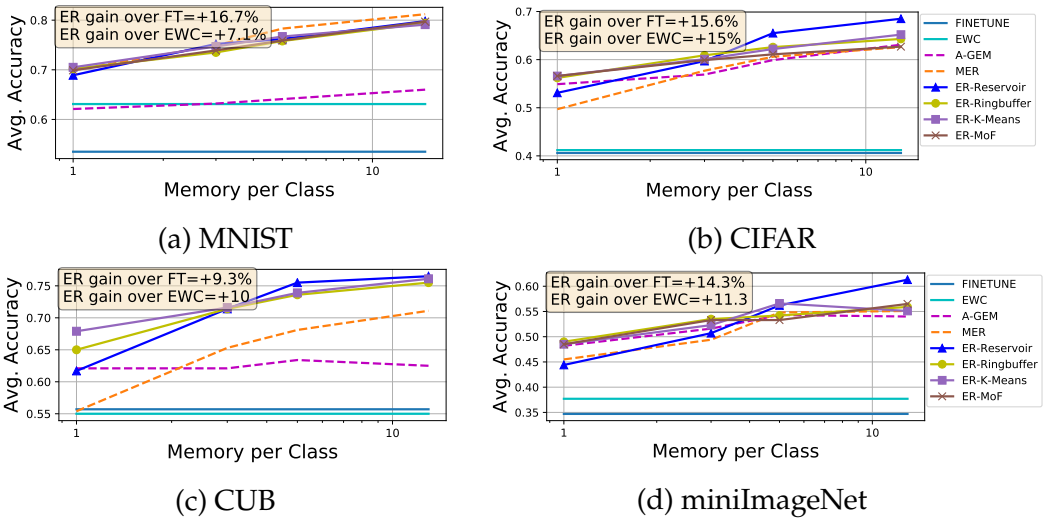
- A-GEM [30], a model that uses episodic memory as an optimization constraint to avoid catastrophic forgetting. Since GEM [87] and A-GEM have similar performance, we only consider the latter in our experiments due to its computational efficiency.

- MER [111], a model that also leverages an episodic memory and uses a loss that approximates the dot products of the gradients of current and previous tasks to avoid forgetting. To make the experimental setting more comparable (in terms of SGD updates) to the other methods, we set the number of inner gradient steps to 1 for each outer Reptile [99] meta-update with the mini-batch size of 10.

5.5.4 Results

In the first experiment, we measured average accuracy at the end of the learning experience on $\mathcal{D}^{EV}$ as a function of the size of the memory (detailed numerical results are provided in Appendix tables 5.3, 5.4, 5.5 and 5.6). From the results in figure 5.1, we can make several observations.

First, methods using *ER greatly outperform* not only the baseline approaches that do not have episodic memory (FINETUNE and EWC) but also state-of-the-art approaches relying on episodic memory of the same size (A-GEM and MER). Moreover, the ER variants outperform even when the episodic memory is very small. For instance, on CIFAR the gain over A-GEM brought by ER is 1.7% when



(a) MNIST (b) CIFAR

(c) CUB (d) miniImageNet

Figure 5.1: Average accuracy as a function of episodic memory size. The box shows the gain in average accuracy of ER-RINGBUFFER over FINETUNE and EWC baselines when only 1 sample per class is used. The performance is averaged over 5 runs. Uncertainty estimates are provided in Appendix tables 5.3, 5.4, 5.5 and 5.6.

the memory only stores 1 example per class, and more than 5% when the memory stores 13 examples per class. This finding might seem quite surprising as repetitive training on a very small episodic memory may potentially lead to overfitting on the examples stored in the memory. We will investigate this finding in more depth in section 5.5.5. In the same setting, the gain compared to methods that do not use memory (FINETUNE and EWC) is 15% and about 28% when using a single example per class and 13 examples per class, respectively. Second and not surprisingly, average accuracy increases with the memory size, and does not saturate at 13 examples per class which is our self-imposed limit.

Third, experience replay based on reservoir sampling works the best across the board except when the memory size is very small (less than 3 examples per class). Empirically we observed that as more and more tasks arrive and the size of the memory per class shrinks, reservoir sampling often ends up evicting some of the earlier classes from the memory, thereby inducing higher forgetting.

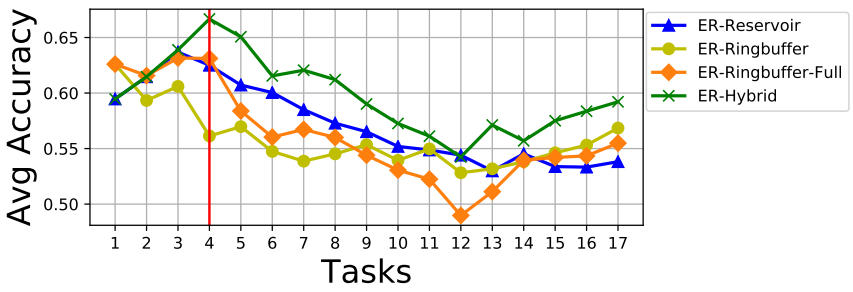


Figure 5.2: Evolution of average accuracy (A_k) as new tasks are learned in Split CIFAR. The memory has only 85 slots (in average 1 slot per class). The vertical bar marks where the hybrid approach switches from reservoir to ring buffer strategy. The hybrid approach works better than both reservoir (once more tasks arrive) and ring buffer (initially, when the memory is otherwise not well utilized). The orange curve is a variant of ring buffer that utilizes the full memory at all times, by reducing the ring buffer size of observed classes as new classes arrive. Overall, the proposed hybrid approach works at least as good as the other approaches throughout the whole learning experience. (Averaged over 3 runs).

Fourth, when the memory is tiny, sampling methods that by construction guarantee a balanced number of samples per class, work the best (even better than reservoir sampling). All methods that have this property, ring buffer, k-Means and Mean of Features, have a rather similar performance which is substantially better than that of the reservoir sampling. For instance, on CIFAR, with one example per class in the memory, ER with reservoir sampling is 3.5% worse than ER K-Means, but ER K-Means, ER Ring Buffer and ER MoF are all within 0.5% from each other (see Appendix table 5.4 for numerical values). These findings are further confirmed by looking at the evolution of the average accuracy (Appendix figure 5.5 left) as new tasks arrive when the memory can store at most one example per class.

The better performance of strategies like ring buffer for tiny episodic memories, and reservoir sampling for bigger episodic memories, suggests a *hybrid* approach, whereby the writing strategy relies on reservoir sampling till some classes have too few samples stored in the memory. At that point, the writing strategy switches to the ring buffer scheme which guarantees a minimum number of examples for each

Methods	Forgetting			
	MNIST	**CIFAR**	**CUB**	**miniImageNet**
FINETUNE	0.29	0.27	0.13	0.26
EWC	0.18	0.27	0.14	0.21
A-GEM	0.21	0.14	0.09	0.13
MER	0.14	0.19	0.10	0.15
ER-RINGBUFFER (ours)	**0.12**	**0.13**	**0.03**	**0.12**

Table 5.1: Forgetting when using a tiny episodic memory of single example per class.

Methods	Training Time [s]	
	CIFAR	**CUB**
FINETUNE	87	194
EWC	159	235
A-GEM	230	510
MER	755	277
ER-RINGBUFFER (ours)	116	255

Table 5.2: Learning Time on $\mathcal{D}^{EV}$ [s]

class. For instance, in the experiment of figure 5.2 the memory budget consists of only 85 memory slots, an average of 1 sample per class by the end of the learning experience (as there are 17 tasks and 5 classes per task). The learner switches from reservoir sampling to ring buffer once it observes that any of the classes seen in the past has only one sample left in the memory. When the switch happens (marked by a red vertical line in the figure), the learner only keeps randomly picked $\min(n, \frac{|\mathcal{M}|}{K})$ examples per class in the memory, where n is the number of examples of class c in the memory and K are the total number of classes observed so far. The overwriting happens opportunistically, removing examples from over-represented classes as new classes are observed. Figure 5.2 shows that when the number of tasks is small, the hybrid version enjoys the high accuracy of reservoir sampling. As more tasks arrive and the memory per task shrinks, the hybrid scheme achieves superior performance than reservoir (and at least similar to ring buffer).

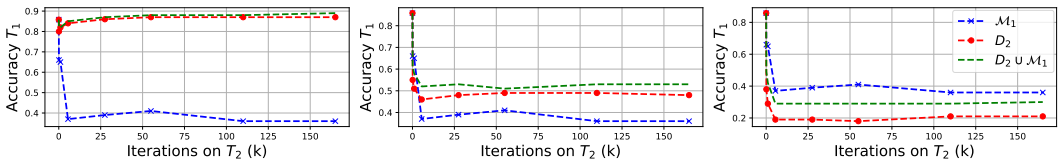
Finally, experience replay methods are not only outperforming all other approaches in terms of accuracy (and lower forgetting as reported in table 5.1), but also in terms of compute time. Table 5.2 reports training time on both Split CIFAR and Split CUB, using ring buffer as a use case since all other ER methods have the same computational complexity. We observe that ER adds only a slight overhead compared to the finetuning baseline, but it is much cheaper than stronger baseline methods like A-GEM and MER.

5.5.5 Analysis

The strong performance of experience replay methods which directly learn using the examples stored in the small episodic memory may be surprising. In fact, Lopez-Paz and Ranzato [87] discounted this repetitive training on the memory option by saying: "Obviously, minimizing the loss at the current example together with [the loss on the episodic memory] results in overfitting to the examples stored in [the memory]". How can the repeated training over the same very small handful of examples possibly generalize?

To investigate this matter we conducted an additional experiment. For simplicity, we consider only two tasks, T_1 and T_2, and study the generalization performance on T_1 as we train on T_2. We denote by $\mathcal{D}_2$ the training set of T_2 and by $\mathcal{M}_1$ the memory storing examples from T_1's training set. Our hypothesis is that although direct training on the examples in $\mathcal{M}_1$ (in addition to those coming from $\mathcal{D}_2$) does indeed lead to strong memorization of $\mathcal{M}_1$ (as measured by nearly zero cross-entropy loss on $\mathcal{M}_1$), such training is still overall beneficial in terms of generalization on the original task T_1 because the joint learning with the examples of the current task T_2 acts as a strong, albeit implicit and data-dependent, regularizer for T_1.

To validate this hypothesis, we consider the MNIST Rotations dataset [87], where each task has digits rotated by a certain degree, a setting that enables fine



(a) 20° rotation (b) 40° rotation (c) 60° rotation

Figure 5.3: Analysis on MNIST Rotation: Test accuracy on Task 1 as a function of the training iterations over Task 2. The blue curves are the accuracy when the model is trained using only $\mathcal{M}_1$. The red curves are the accuracy when the model is trained using only $\mathcal{D}_2$, the training set of Task 2. The green curves are the accuracy when in addition to $\mathcal{D}_2$, the model uses the memory from Task 1, $\mathcal{M}_1$ (experience replay). (Averaged over 3 runs).

control over the relatedness between the tasks. The architecture is the same as for Permuted MNIST, with only 10 memory slots, one for each class of T_1. First, we verified that the loss on $\mathcal{M}_1$ quickly drops to nearly 0 as the model is trained using both $\mathcal{M}_1$ and $\mathcal{D}_2$. As expected, the model achieves a perfect performance on the examples in the memory, which is not true for methods like A-GEM which make less direct use of the memory (see Appendix table 5.7). We then verified that only training on $\mathcal{M}_1$ without $\mathcal{D}_2$, yields strong overfitting to the examples in the memory and poor generalization performance, with a mere average accuracy of 40% on T_1 from the initial 85% which was obtained just after training on T_1. If we only train on $\mathcal{D}_2$ without using $\mathcal{M}_1$ (same as FINETUNE baseline), we also observed overfitting to $\mathcal{D}_2$ as long as T_2 and T_1 are sufficiently unrelated, figure 5.3(b) and figure 5.3(c).

When the two tasks are closely related instead (difference of rotation angles less than 20 degrees), we observe that even without the memory, generalization on T_1 improves as we train on T_2 because of positive transfer from the related task, see red curve in figure 5.3(a). However, when we train on both $\mathcal{D}_2$ and $\mathcal{M}_1$, generalization on T_1 is better than FINETUNE baseline, *i.e.*, training with $\mathcal{D}_2$ only, regardless of the degree of relatedness between the two tasks, as shown by the green curves in figure 5.3.

These findings suggest that while the model essentially memorizes the examples in the memory, this does not necessarily have a detrimental effect in terms of generalization as long as such learning is performed in conjunction with the examples of T_2. Moreover, there are two major axes controlling this regularizer: the number of examples in T_2 and the relatedness between the tasks. The former sets the strength of the regularizer. The latter, as measured by the accuracy on T_1 when training only on $\mathcal{D}_2$, controls its effectiveness. When T_1 and T_2 are closely related, figure 5.3(a), training on $\mathcal{D}_2$ prevents overfitting to $\mathcal{M}_1$ by providing a data-dependent regularization that, even by itself, produces positive transfer. When T_1 and T_2 are somewhat related, figure 5.3(b), training on $\mathcal{D}_2$ still improves generalization on T_1 albeit to a much lesser extent. However, when the tasks are almost adversarial to each other as an upside down 2 may look like a 5, the resulting regularization becomes even harmful, figure 5.3(c). In this case, accuracy drops from 40% (training only on $\mathcal{M}_1$) to 30% (training on both $\mathcal{M}_1$ and $\mathcal{D}_2$).

One remaining question related to generalization is how ER relates to A-GEM [30] and whether A-GEM overfits even less? The answer is positive. As shown in Appendix table 5.7, A-GEM's accuracy on the memory examples does not reach 100% even after having processed 1000 samples. Interestingly, accuracy on the training set is lower than ER suggesting that the more constrained weight updates of A-GEM make it actually underfit. This underfitting prevents A-GEM from reaping the full regularization benefits brought by training on the data of subsequent tasks.

5.6 Conclusion

In this chapter we studied ER methods for supervised CL tasks. Our empirical analysis on several benchmark streams of data shows that ER methods even with

a tiny episodic memory offer a very large performance boost at a very marginal increase of computational cost compared to the finetuning baseline. We also studied various ways to populate the memory and proposed a hybrid approach that strikes a good trade-off between randomizing the examples in the memory while keeping enough representatives for each class.

Our study also sheds light into a very interesting phenomenon: memorization (zero cross-entropy loss) of tiny memories is useful for generalization because training on subsequent tasks acts like a data dependent regularizer. Overall, we hope the CL community will adopt experience replay methods as a baseline, given their strong empirical performance, efficiency and simplicity of implementation.

There are several avenues of future work. For instance, we would like to investigate what are the optimal inputs that best mitigate expected forgetting and optimal strategies to remove samples from the memory when it is entirely filled up.

Appendix

In section 5.A, we provide algorithms for different memory update strategies described in section 5.4 of the main chapter 5. The detailed results of the experiments which were used to generate figure 5.1 and table 5.1 in the main chapter 5 are provided in section 5.B. The analysis conducted in section 5.5.5 of the main chapter 5 is further described in section 5.C. Finally, in section 5.D, we list the hyper-parameters used for each of the baselines across all the datasets.

5.A Memory Update Algorithms

Here we provide the algorithms to write into memory as discussed in section 5.4 of the main chapter 5.

Algorithm 4 Reservoir sampling update. mem_sz is the number of examples the memory can store, t is the task id, n is the number of examples observed so far in the data stream, and B is the input mini-batch.

procedure UpdateMemory(mem_sz, t, n, B)
  $j \leftarrow 0$
  for $(\mathbf{x}, y)$ in B **do**
    $M \leftarrow |\mathcal{M}|$ ▷ Number of samples currently stored in the memory
    if $M <$ mem_sz **then**
      $\mathcal{M}$.append$(\mathbf{x}, y, t)$
    else
      $i =$ randint$(0, n + j)$
      if $i <$ mem_sz **then**
        $\mathcal{M}[i] \leftarrow (\mathbf{x}, y, t)$ ▷ Overwrite memory slot.
      end if
    end if
    $j \leftarrow j + 1$
  end for
  return $\mathcal{M}$
end procedure

Algorithm 5 Ring buffer.

procedure UPDATEMEMORY(mem_sz, t, n, B)
  for $(\mathbf{x}, y)$ in B **do**
    # Assume FIFO stacks $\mathcal{M}[t][y]$ of fixed size are already initialized
    $\mathcal{M}[t][y]$.append($\mathbf{x}$)
  end for
  return $\mathcal{M}$
end procedure

Algorithm 6 K-Means. Memory is populated using samples closest (in feature space) to sequential K-Means centroids.

procedure UPDATEMEMORY(mem_sz, t, n, B)
  # Assume array $\mathcal{M}[t][y]$ of fixed size is already initialized
  # Assume K centroids c_j are already initialized
  # Assume cluster counters n_j are already initialized to 0
  for $(\mathbf{x}, y)$ in B **do**
    $j \leftarrow \arg\min_{j \in \{1,\cdots,K\}} ||\phi_\theta(\mathbf{x}) - c_j||$
    $n_j \leftarrow n_j + 1$
    $c_j \leftarrow c_j + \frac{1}{n_j} * (\phi_\theta(\mathbf{x}) - c_j)$
    $d = ||\phi_\theta(\mathbf{x}) - c_j||$
    if $d < \mathcal{M}[t][y][j]$.get_dst() **then** ▷ Store the current example if it is closer to the centroid
      $\mathcal{M}[t][y][j] \leftarrow (\mathbf{x}, d)$
    end if
  end for
  return $\mathcal{M}$
end procedure

Algorithm 7 Mean of Features. Store examples that are closest to the running average feature vector.

procedure UPDATEMEMORY(mem_sz, t, n, B)
  # Assume heaps $\mathcal{M}[t][y]$ of fixed size are already initialized
  # Assume average features $f[t][y]$ are already initialized
  # Assume moving average decay hyper-parameter (α) is given
  for $(\mathbf{x}, y)$ in B **do**
    $f[t][y] \leftarrow \alpha * f[t][y] + (1 - \alpha) * \phi_\theta(\mathbf{x})$
    $d = ||\phi_\theta(\mathbf{x}) - f[t][y]||$
    if $\mathcal{M}[t][y]$.find_max() $> d$ **then** ▷ Store the current example if it is closer to the center
      $\mathcal{M}[t][y]$.delete_max()
      $\mathcal{M}[t][y]$.insert($\mathbf{x}; d$)
    end if
  end for
  return $\mathcal{M}$
end procedure

5.B Detailed Results

Here we describe the detailed results used to generate the figure 5.1 in the main chapter 5. In addition we also report the forgetting metric equation 5.3. Note that the MULTI-TASK baseline does not follow the definition of continual learning as it keeps the dataset of all the tasks around at every step.

Table 5.3: Permuted MNIST: Performance (average accuracy (left column) and forgetting (right column)) for different number of samples per class. The average accuracy numbers from the this table are used to generate figure 5.1 in section 5.5.4 of the main chapter 5.

Methods	**Episodic Memory (Samples Per Class)**							
	Average Accuracy [A_T(%)]				Forgetting [F_T]			
	1	3	5	15	1	3	5	15
A-GEM	62.1 (± 1.39)	63.2 (± 1.47)	64.1 (± 0.74)	66.0 (± 1.78)	0.21 (± 0.01)	0.20 (± 0.01)	0.19 (± 0.01)	0.17 (± 0.02)
MER	69.9 (± 0.40)	74.9 (± 0.49)	78.3 (± 0.19)	81.2 (± 0.28)	0.14 (± 0.01)	0.09 (± 0.01)	0.06 (± 0.01)	0.03 (± 0.01)
ER-RINGBUFFER	70.2 (± 0.56)	73.5 (± 0.43)	75.8 (± 0.24)	79.4 (± 0.43)	0.12 (± 0.01)	0.09 (± 0.01)	0.07 (± 0.01)	0.04 (± 0.01)
ER-MOF	69.9 (± 0.68)	73.9 (± 0.64)	75.9 (± 0.21)	79.7 (± 0.19)	0.13 (± 0.01)	0.09 (± 0.01)	0.07 (± 0.01)	0.04 (± 0.01)
ER-K-MEANS	70.5 (± 0.42)	74.7 (± 0.62)	76.7 (± 0.51)	79.1 (± 0.32)	0.12 (± 0.01)	0.08 (± 0.01)	0.06 (± 0.01)	0.04 (± 0.01)
ER-RESERVOIR	68.9 (± 0.89)	75.2 (± 0.33)	76.2 (± 0.38)	79.8 (± 0.26)	0.15 (± 0.01)	0.08 (± 0.01)	0.07 (± 0.01)	0.04 (± 0.01)
FINETUNE	53.5 (± 1.46)	-	-	-	0.29 (± 0.01)	-	-	
EWC	63.1 (± 1.40)	-	-	-	0.18 (± 0.01)	-	-	
MULTI-TASK	83				-			

Table 5.4: Split CIFAR: Performance (average accuracy (left column) and forgetting (right column)) for different number of samples per class. The average accuracy numbers from the this table are used to generate figure 5.1 in section 5.5.4 of the main chapter 5.

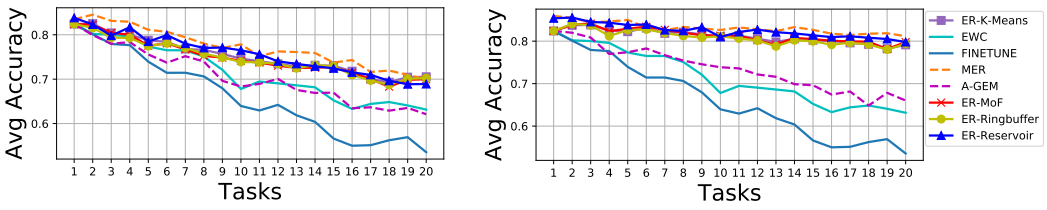
Methods	**Episodic Memory (Samples Per Class)**							
	Average Accuracy [A_T(%)]				Forgetting [F_T]			
	1	3	5	13	1	3	5	13
A-GEM	54.9 (± 2.92)	56.9 (± 3.45)	59.9 (± 2.64)	63.1 (± 1.24)	0.14 (± 0.03)	0.13 (± 0.03)	0.10 (± 0.02)	0.07 (± 0.01)
MER	49.7 (± 2.97)	57.7 (± 2.59)	60.6 (± 2.09)	62.6 (± 1.48)	0.19 (± 0.03)	0.11 (± 0.01)	0.09 (± 0.02)	0.07 (± 0.01)
ER-RINGBUFFER	56.2 (± 1.93)	60.9 (± 1.44)	62.6 (± 1.77)	64.3 (± 1.84)	0.13 (± 0.01)	0.09 (± 0.01)	0.08 (± 0.02)	0.06 (± 0.01)
ER-MOF	56.6 (± 2.09)	59.9 (± 1.25)	61.1 (± 1.62)	62.7 (± 0.63)	0.12 (± 0.01)	0.10 (± 0.01)	0.08 (± 0.01)	0.07 (± 0.01)
ER-K-MEANS	56.6 (± 1.40)	60.1 (± 1.41)	62.2 (± 1.20)	65.2 (± 1.81)	0.13 (± 0.01)	0.09 (± 0.01)	0.07 (± 0.01)	0.04 (± 0.01)
ER-RESERVOIR	53.1 (± 2.66)	59.7 (± 3.87)	65.5 (± 1.99)	68.5 (± 0.65)	0.19 (± 0.02)	0.12 (± 0.03)	0.09 (± 0.02)	0.05 (± 0.01)
FINETUNE	40.6 (± 3.83)	-	-	-	0.27 (± 0.04)	-	-	
EWC	41.2 (± 2.67)	-	-	-	0.27 (± 0.02)	-	-	
MULTI-TASK	68.3				-			

Table 5.5: miniImageNet: Performance (average accuracy (left column) and forgetting (right column)) for different number of samples per class. The average accuracy numbers from the this table are used to generate figure 5.1 in section 5.5.4 of the main chapter 5.

Methods	**Episodic Memory (Samples Per Class)**							
	Average Accuracy [A_T(%)]				Forgetting [F_T]			
	1	3	5	13	1	3	5	13
A-GEM	48.2 ($\pm$ 2.49)	51.6 ($\pm$ 2.69)	54.3 ($\pm$ 1.56)	54 ($\pm$ 3.63)	0.13 ($\pm$ 0.02)	0.10 ($\pm$ 0.02)	0.08 ($\pm$ 0.01)	0.09 ($\pm$ 0.03)
MER	45.5 ($\pm$ 1.49)	49.4 ($\pm$ 3.43)	54.8 ($\pm$ 1.79)	55.1 ($\pm$ 2.91)	0.15 ($\pm$ 0.01)	0.12 ($\pm$ 0.02)	0.07 ($\pm$ 0.01)	0.07 ($\pm$ 0.01)
ER-RINGBUFFER	49.0 ($\pm$ 2.61)	53.5 ($\pm$ 1.42)	54.2 ($\pm$ 3.23)	55.9 ($\pm$ 4.05)	0.12 ($\pm$ 0.02)	0.07 ($\pm$ 0.02)	0.08 ($\pm$ 0.02)	0.06 ($\pm$ 0.03)
ER-MOF	48.5 ($\pm$ 1.72)	53.3 ($\pm$ 2.80)	53.3 ($\pm$ 3.11)	56.5 ($\pm$ 1.92)	0.12 ($\pm$ 0.01)	0.08 ($\pm$ 0.01)	0.08 ($\pm$ 0.02)	0.05 ($\pm$ 0.02)
ER-K-MEANS	48.5 ($\pm$ 0.35)	52.3 ($\pm$ 3.12)	56.6 ($\pm$ 2.48)	55.1 ($\pm$ 1.86)	0.12 ($\pm$ 0.02)	0.09 ($\pm$ 0.02)	0.06 ($\pm$ 0.01)	0.06 ($\pm$ 0.01)
ER-RESERVOIR	44.4 ($\pm$ 3.22)	50.7 ($\pm$ 3.36)	56.2 ($\pm$ 4.12)	61.3 ($\pm$ 6.72)	0.17 ($\pm$ 0.02)	0.12 ($\pm$ 0.03)	0.07 ($\pm$ 0.04)	0.04 ($\pm$ 0.06)
FINETUNE	34.7 ($\pm$ 2.69)	-	-	-	0.26 ($\pm$ 0.03)	-	-	
EWC	37.7 ($\pm$ 3.29)	-	-	-	0.21 ($\pm$ 0.03)	-	-	
MULTI-TASK	62.4				-			

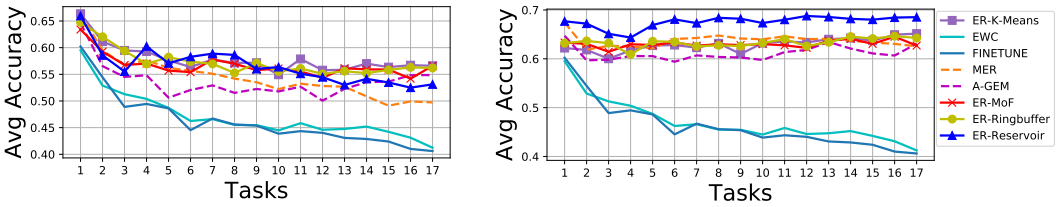
Table 5.6: CUB: Performance (average accuracy (left column) and forgetting (right column)) for different number of samples per class. The average accuracy numbers from the this table are used to generate figure 5.1 in section 5.5.4 of the main chapter 5.

Methods	**Episodic Memory (Samples Per Class)**							
	Average Accuracy [A_T(%)]				Forgetting [F_T]			
	1	3	5	10	1	3	5	10
A-GEM	62.1 ($\pm$ 1.28)	62.1 ($\pm$ 1.87)	63.4 ($\pm$ 2.33)	62.5 ($\pm$ 2.34)	0.09 ($\pm$ 0.01)	0.08 ($\pm$ 0.02)	0.07 ($\pm$ 0.01)	0.08 ($\pm$ 0.02)
MER	55.4 ($\pm$ 1.03)	65.3 ($\pm$ 1.68)	68.1 ($\pm$ 1.61)	71.1 ($\pm$ 0.93)	0.10 ($\pm$ 0.01)	0.04 ($\pm$ 0.01)	0.03 ($\pm$ 0.01)	0.03 ($\pm$ 0.01)
ER-RINGBUFFER	65.0 ($\pm$ 0.96)	71.4 ($\pm$ 1.53)	73.6 ($\pm$ 1.57)	75.5 ($\pm$ 1.84)	0.03 ($\pm$ 0.01)	0.01 ($\pm$ 0.01)	0.01 ($\pm$ 0.01)	0.02 ($\pm$ 0.01)
ER-K-MEANS	67.9 ($\pm$ 0.87)	71.6 ($\pm$ 1.56)	73.9 ($\pm$ 2.01)	76.1 ($\pm$ 1.74)	0.02 ($\pm$ 0.01)	0.02 ($\pm$ 0.01)	0.02 ($\pm$ 0.01)	0.01 ($\pm$ 0.01)
ER-RESERVOIR	61.7 ($\pm$ 0.62)	71.4 ($\pm$ 2.57)	75.5 ($\pm$ 1.92)	76.5 ($\pm$ 1.56)	0.09 ($\pm$ 0.01)	0.04 ($\pm$ 0.01)	0.02 ($\pm$ 0.01)	0.03 ($\pm$ 0.02)
FINETUNE	55.7 ($\pm$ 2.22)	-	-	-	0.13 ($\pm$ 0.03)	-	-	
EWC	55.0 ($\pm$ 2.34)	-	-	-	0.14 ($\pm$ 0.02)	-	-	
MULTI-TASK	65.6				-			



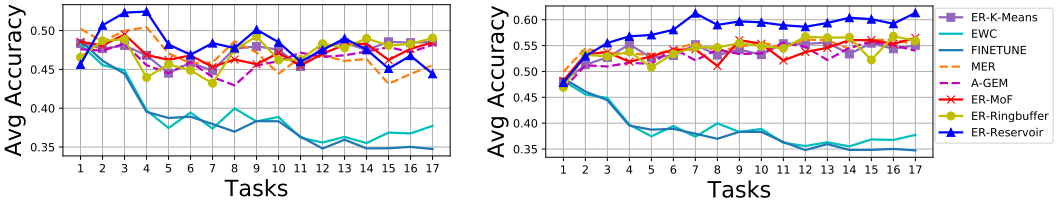
(a) 1 Sample (b) 15 Samples

Figure 5.4: MNIST: Evolution of average accuracy (A_k) as new tasks are learned when '1' and '15' samples per class are used.



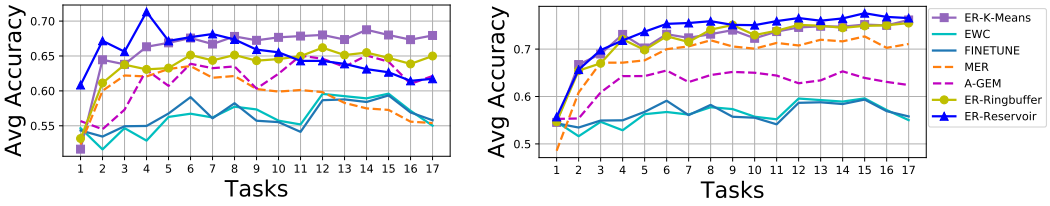
(a) 1 Sample (b) 13 Samples

Figure 5.5: CIFAR: Evolution of average accuracy (A_k) as new tasks are learned when '1' and '13' samples per class. The performance is averaged over 5 runs. Uncertainty estimates are provided in tables 5.3, 5.4, 5.5 and 5.6.



(a) 1 Sample (b) 13 Samples

Figure 5.6: miniImageNet: Evolution of average accuracy (A_k) as new tasks are learned when '1' and '13' samples per class are used.



(a) 1 Sample (b) 10 Samples

Figure 5.7: CUB: Evolution of average accuracy (A_k) as new tasks are learned when '1' and '10' samples per class are used.

5.C Further Analysis

Table 5.7: MNIST Rotation: Performance of task 1 after training on task 2.

Task 2 Samples	**Rotation Angle**											
	10						**90**					
	ER-RINGBUFFER			A-GEM			ER-RINGBUFFER			A-GEM		
	Train	Mem	Test	Train	Mem	Test	Train	Mem	Test	Train	Mem	Test
1000	85.6	1	86.2	81.5	86.6	82.5	68.7	1	69.4	51.7	73.3	52.1
20000	91.4	1	91.6	91.4	1	91.5	32.7	1	33.4	31.6	1	33.0

In table 5.7, we provide train, memory and test set performance on both the ER-RINGBUFFER and A-GEM with two different configurations of tasks; similar tasks (10rotation), dissimilar tasks (90rotation). It can be seen from the table, and as argued in the section 5.5.5 of the main chapter 5 that ER-RINGBUFFER always achieves the perfect performance on the memory. To achieve the same effect with A-GEM one has to train for more iterations.

5.D Hyper-parameter Selection

Table 5.8: Hyper-parameters selection on the four benchmark datasets. 'lr' is the learning rate, 'λ' is the synaptic strength for EWC, 'γ' is the with in batch meta-learning rate for MER, 's' is current example learning rate multiplier for MER.

Methods	**MNIST**	**CIFAR**	**CUB**	**miniImageNet**
FINETUNE	lr (0.1)	lr (0.03)	lr (0.03)	lr (0.03)
EWC	lr (0.1), λ (10)	lr (0.03), λ (10)	lr (0.03), λ (10)	lr (0.03), λ (10)
A-GEM	lr (0.1)	lr (0.03)	lr (0.03)	lr (0.03)
MER	lr (0.03), γ (0.1), s (10)	lr (0.03), γ (0.1), s (5)	lr (0.1), γ (0.1), s (5)	lr (0.03), γ (0.1), s (5)
ER-RESERVOIR	lr (0.1)	lr (0.1)	lr (0.03)	lr (0.1)
ER-[OTHERS]	lr (0.1)	lr (0.03)	lr (0.03)	lr (0.03)

Chapter 6

Synthesizing and Anchoring Memory Samples for Continual Learning

Abstract

In continual learning, the learner faces a stream of data whose distribution changes over time. Modern neural networks are known to suffer under this setting, as they quickly forget previously acquired knowledge. To address such catastrophic forgetting, many continual learning methods implement different types of *experience replay*, re-learning on past data stored in a small buffer known as episodic memory. In this chapter, we complement experience replay with a new objective that we call "anchoring", where the learner uses bilevel optimization to update its knowledge on the current task, while keeping intact the predictions on some *anchor points* of past tasks. These anchor points are learned using gradient-based optimization to maximize forgetting, which is approximated by fine-tuning the currently trained model on the episodic memory of past tasks. Experiments on several supervised learning benchmarks for continual learning demonstrate that our approach im-

proves the standard experience replay in terms of both accuracy and forgetting metrics and for various sizes of episodic memories.

6.1 Introduction

We study the problem of *continual learning*, where a machine learning model experiences a sequence of tasks. Each of these tasks is presented as a stream of input-output pairs, where each pair is drawn identically and independently (iid) from the corresponding task probability distribution. Since the length of the learning experience is not specified a-priori, the learner can only assume a *single pass over the data* and, due to space constraints, store nothing but a few examples in a small *episodic memory*. At all times during the lifetime of the model, predictions on examples from all tasks may be requested. Addressing continual learning is an important research problem, since it would enable the community to move past the assumption of "identically and independently distributed data", and allow a better deployment of machine learning in-the-wild. However, continual learning presents one major challenge, *catastrophic forgetting* [93]. That is, as the learner experiences new tasks, it quickly forgets previously acquired knowledge. This is a hindrance specially for state-of-the-art deep learning models, where all parameters are updated after observing each example.

Continual learning has received increasing attention from the scientific community during the last decade. The state-of-the-art in algorithms for continual learning fall into three categories. First, *regularization-based* approaches reduce forgetting by restricting the updates in parameters that were important for previous tasks [67, 110, 3, 29, 98]. However, when the number of tasks are large, the regularization of past tasks becomes obsolete, leading to representation drift [141]. Second, *modular approaches* [118, 80] add new modules to the learner as new tasks are

learned. While modular approaches overcome forgetting by design, the memory complexity of these approaches scales with the number of tasks. Third, *memory-based* methods [87, 52, 60, 111, 30] store a few examples from past tasks in an "episodic memory", to be revisited when training for a new task. Contrary to modular approaches, memory-based methods add a very small memory overhead for each new task. Memory-based methods are the reigning state-of-the-art, but their performance remains a far-cry from a simple oracle accessing all the data at once, hence turning the continual learning experience back into a normal supervised learning task. Despite intense research efforts, such gap in performance renders the problem of continual learning an open research question.

Contribution We propose Hindsight Anchor Learning (HAL), a continual learning approach to improve the performance of memory-based continual learning algorithms. HAL leverages bilevel optimization to regularize the training objective with one representational point per class per task, called *anchors*. These anchors are constructed via gradient ascent in the image space, by maximizing one approximation to the forgetting loss for the current task throughout the entire continual learning experience. We estimate the amount of forgetting that the learner would suffer on these anchors if it were to be trained on future tasks in *hindsight*: that is, by measuring forgetting on a temporary predictor that has been fine-tuned on the episodic memories of past tasks. Anchors learned in such a way lie close to the classifier's decision boundary, as visualized in figure 6.2. Since points near the decision boundary are easiest to forget when updating the learner on future tasks, keeping predictions invariant on such anchors preserves performance at previous tasks effectively. In sum, the overall parameter update of HAL uses nested optimization to minimize the loss on the current mini-batch, while keeping the predictions at all anchors invariant.

Results We compare HAL to EWC [67], ICARL [110], VCL [98], AGEM [30], experience replay [52, 111], and MER [111], across four commonly used benchmarks in supervised continual learning (MNIST permutations, MNIST rotations, split CIFAR-100, and split miniImageNet). In these experiments, HAL achieves state-of-the-art performance, improving accuracy by upto 7.5% and reducing forgetting by almost 23% in the best case. We show that these results hold for various sizes of episodic memories (between 1 and 5 examples per class per task).

We now begin our exposition by reviewing the continual learning setup. The rest of the chapter then presents our new algorithm HAL (section 6.3), showcases its empirical performance (section 6.4), surveys the related literature (section 6.5), and offers some concluding remarks (section 6.6).

6.2 Continual learning setup

In continual learning, we experience a stream of data triplets (x_i, y_i, t_i) containing an input x_i, a target y_i, and a task identifier $t_i \in \mathcal{T} = \{1, \ldots, T\}$. Each input-target pair $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}_{t_i}$ is an identical and independently distributed example drawn from some unknown distribution $P_{t_i}(X, Y)$, representing the t_i-th learning task. We assume that the tasks are experienced in order ($t_i \leq t_j$ for all $i \leq j$), and that the total number of tasks T is not known a priori. Under this setup, our goal is to estimate a predictor $f_\theta = (w \circ \phi) : \mathcal{X} \times \mathcal{T} \to \mathcal{Y}$, parameterized by $\theta \in \mathbb{R}^P$, and composed of a feature extractor $\phi : \mathcal{X} \to \mathcal{H}$ and a classifier $w : \mathcal{H} \to \mathcal{Y}$, that minimizes the multi-task error

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{(x,y)\sim P_t} \left[\ell(f(x, t), y) \right], \tag{6.1}$$

where $\mathcal{Y} = \cup_{t\in\mathcal{T}} \mathcal{Y}_t$, and $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function.

Inspired by prior literature in continual learning [87, 52, 111, 30], we consider

streams of data that are *experienced only once*. Therefore, the learner cannot revisit any but a small amount of data triplets chosen to be stored in a small episodic memory $\mathcal{M}$. More specifically, we consider tiny "ring" episodic memories, which contain the last m observed examples per class for each of the experienced tasks, where $m \in \{1, 3, 5\}$. That is, considering as variables the number of experienced tasks t and examples n, we study continual learning algorithms with a $O(t)$ memory footprint.

Following Lopez-Paz and Ranzato [87] and Chaudhry et al. [29], we monitor two statistics to evaluate the quality of continual learning algorithms: *final average accuracy*, and *final maximum forgetting*. First, the final average accuracy of a predictor is defined as

$$\text{Accuracy} = \frac{1}{T}\sum_{j=1}^{T} a_{T,j}, \tag{6.2}$$

where $a_{i,j}$ denotes the test accuracy on task j after the model has finished experiencing task i. That is, the final average accuracy measures the test performance of the model at every task after the continual learning experience has finished. Second, the final maximum forgetting is defined as

$$\text{Forgetting} = \frac{1}{T-1}\sum_{j=1}^{T-1} \max_{l\in\{1,\dots,T-1\}} (a_{l,j} - a_{T,j}), \tag{6.3}$$

that is, the decrease in performance at each of the tasks between their peak accuracy and their accuracy after the continual learning experience has finished.

Finally, following Chaudhry et al. [30], we use the first $k < T$ tasks to cross-validate the hyper-parameters of each of the considered continual learning algorithms. These first k tasks are not considered when computing the final average accuracy and maximum forgetting metrics.

6.3 Hindsight Anchor Learning (HAL)

The current state-of-the-art algorithms for continual learning are based on experience replay [52, 111, 31]. These methods update the model f_θ while storing a small amount of past observed triplets in an episodic memory $\mathcal{M} = \{(x', y', t')\}$. For a new mini-batch of observations $\mathcal{B} := \{(x, y, t)\}$ from task t, the learner samples a mini-batch $\mathcal{B}_\mathcal{M}$ from $\mathcal{M}$ at random, and employ the rule $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \, \ell(\mathcal{B} \cup \mathcal{B}_\mathcal{M})$ to update its parameters, where

$$\ell(\mathcal{A}) = \frac{1}{|\mathcal{A}|} \sum_{(x,y,t)\in\mathcal{A}} \ell(f_\theta(x, t), y) \tag{6.4}$$

denotes the average loss across a collection of triplets $\mathcal{A} = \mathcal{B} \cup \mathcal{B}_\mathcal{M}$. In general, $\mathcal{B}_\mathcal{M}$ is constructed to have the same size as $\mathcal{B}$, but it can be smaller if the episodic memory $\mathcal{M}$ does not yet contain enough samples.

The episodic memory $\mathcal{M}$ reminds the predictor about how to perform at past tasks using only a small amount of data. As such, the behaviour of the predictor on past tasks outside the data stored in $\mathcal{M}$ is not guaranteed. Also, since $\mathcal{M}$ is usually very small, the performance of the predictor becomes sensitive to the choice of samples stored in the episodic memory. Because of this reason, we propose to further fix the behaviour of the predictor at a collection of carefully constructed *anchor points* $e_{t'}$, one per class per past task t', at each parameter update.

Let us assume that the anchor points $e_{t'}$ are given —we will see later how to construct them in practice. To constrain the change of the predictor at these anchor points, we propose a two-step parameter update rule:

$$\begin{aligned}
&\tilde{\theta} \leftarrow \theta - \alpha \nabla_\theta \, \ell(\mathcal{B} \cup \mathcal{B}_\mathcal{M}), \\
&\theta \leftarrow \theta - \alpha \nabla_\theta \left(\ell(\mathcal{B} \cup \mathcal{B}_\mathcal{M}) + \lambda \sum_{t'<t} \left(f_\theta(e_{t'}, t') - f_{\tilde{\theta}}(e_{t'}, t') \right)^2 \right).
\end{aligned} \tag{6.5}$$

The first step computes a temporary parameter vector $\tilde{\theta}$ by minimizing the loss at a minibatch from the current task t, and the episodic memory of past tasks (this is the usual experience replay parameter update). The second step employs nested optimization to perform the final update of the parameter θ, which trades-off the minimization of (a) the loss value at the current minibatch and the episodic memory, as well as (b) the change in predictions at the anchor points for all past tasks. The proposed rule not only updates the predictor conservatively, thereby reducing forgetting, but also, as shown in Appendix section 6.A, improves the transfer by maximizing the inner product between the gradients on $\mathcal{B} \cup \mathcal{B}_{\mathcal{M}}$ and anchor points. In this respect, it bears similarity to gradient-based meta-learning approaches [42, 99, 111].

Next, let us discuss how to choose the anchor points, e_t (one per class per task) as to preserve the performance of current task throughout the entire learning experience. Ideally, the anchor points should attempt to minimize the forgetting on the current task as the learner is updated on future tasks. One could achieve this by letting e_t to be an example from the task t that would undergo maximum forgetting during the entire continual learning experience, including past and future tasks. Then, requiring the predictions to remain invariant at such e_t, by using equation 6.5, could effectively reduce forgetting on the current task. Mathematically, the desirable e_t for the label y_t is obtained by maximizing the following *Forgetting loss*:

$$(e_t, y_t) \leftarrow \underset{(x,y)\sim P_t}{\arg\max}\ \ell(f_{\theta_T}(x,t), y_t) - \ell(f_{\theta_t}(x,t), y_t), \tag{6.6}$$

where θ_t is the parameter vector obtained after training on task t and θ_T is the final parameter vector obtained after the entire learning experience. Thus, keeping predictions intact on the pair (e_t, y_t) above can effectively preserve the performance of task t. However, the idealistic equation 6.6 requires access to (a) the entire distribution P_t to compute the maximization, and (b) access to all future distributions

$t' > t$ to compute the final parameter vector θ_T. Both are unrealistic assumptions under the continual learning setup described in section 6.2, as the former requires storing the entire dataset of task t, and the latter needs access to future tasks.

To circumvent (a), we can recast equation 6.6 as an optimization problem and *learn* the desired e_t by initializing it at random and using k gradient ascent updates for a *given label* y_t in the image space ($\mathcal{X} \in \mathbb{R}^D$). The proposed optimization objective is given by:

$$\max_{e_t \in \mathbb{R}^D} \Bigg(\underbrace{\ell(f_{\theta_T}(e_t, t), y_t) - \ell(f_{\theta_t}(e_t, t), y_t)}_{\text{Forgetting loss}} - \gamma \underbrace{(\phi(e_t) - \phi_t)^2}_{\text{Mean embedding loss}} \Bigg), \tag{6.7}$$

where the regularizer, given by mean embedding loss, constrains the search space by trying to push the anchor point embedding towards the mean data embedding. We recall that ϕ denotes the feature extractor of the predictor, and ϕ_t is the neural mean embedding [134] of all observed examples from task t. Since the feature extractor is updated after experiencing each data point, the mean embedding ϕ_t are computed as running averages. That is, after observing a minibatch $\mathcal{B} = \{(x, y, t)\}$ of task t, we update:

$$\phi_t \leftarrow \beta \cdot \phi_t + (1 - \beta) \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \phi(x), \tag{6.8}$$

where ϕ_t is initialized to zero at the beginning of the learning experience. In our experiments, we learn one e_t per class for each task. We fix the y_t to the corresponding class label, and discard ϕ_t after training on task t. Learning e_t in this manner circumvents the requirement of storing the entire distribution P_t for the current task t.

Still, equation 6.7 requires the parameter vector θ_T, to be obtained in the distant future after all learning tasks have been experienced. To waive this impossible

Algorithm 8 Hindsight Anchor Learning (HAL)

Initialize $\theta \sim P(\theta)$ and $\{e_t \sim P(e)\}_{t=1}^{T}$ from normal distributions $P(\theta)$ and $P(e)$, $\mathcal{M} = \{\}$.
Initialize $\mathcal{M} = \{\}$
for $t = 1, \ldots, T$ **do**
    Sample $\mathcal{B}_{\mathcal{M}}$ from $\mathcal{M}$
    Update θ using equation 6.5
    Update ϕ_t using equation 6.8
    Update $\mathcal{M}$ by adding $\mathcal{B}$ in a first-in first-out (FIFO) ring buffer
    Fine-tune on $\mathcal{M}$ to obtain $\theta_{\mathcal{M}}$
    Build e_t using equation 6.9 k times
    Discard ϕ_t
end for
Return θ.

requirement, we propose to *approximate the future by simulating the past*. That is, instead of measuring the forgetting that would happen after the model is trained at future tasks, we measure the forgetting that happens when the model is fine-tuned at past tasks. In this way, we say that forgetting is estimated in *hindsight*, using past experiences. More concretely, after training on task t and obtaining the parameter vector θ_t, we minimize the loss during one epoch on the episodic memory $\mathcal{M}$ to obtain a temporary parameter vector $\theta_{\mathcal{M}}$ that approximates θ_T, and update e_t as:

$$e_t \leftarrow e_t + \alpha \nabla_{e_t} \Bigg(\underbrace{\ell(f_{\theta_{\mathcal{M}}}(e_t, t), y_t) - \ell(f_{\theta_t}(e_t, t), y_t)}_{\text{Hindsight forgetting loss}} - \gamma \underbrace{(\phi(e_t) - \phi_t)^2}_{\text{Mean embedding loss}} \Bigg). \tag{6.9}$$

This completes the description of our proposed algorithm for continual learning, which combines experience replay with anchors learned in hindsight. We call our approach Hindsight Anchor Learning (HAL) and summarize the entire learning process in algorithm 8.

We contrast HAL from another idea Maximally Interfered Retrieval (MIR) [4] in that MIR *selects* a minibatch from an already populated replay buffer at each training step, whereas HAL *writes* to the replay buffer at the end of the training of each task and samples randomly from the replay buffer when training for future

tasks. Further, MIR selects the minibatch by measuring a *one-step* increase in loss incurred by the minibatch whereas HAL uses an approximate forgetting loss equation 6.9 that measures an increase over multiple training steps.

6.4 Experiments

We now report experiments on standard image classification benchmarks for continual learning.

6.4.1 Datasets and tasks

We perform experiments on four supervised classification benchmarks for continual learning. **Permuted MNIST** is a variant of the MNIST dataset of handwritten digits [78] where each task applies a fixed random pixel permutation to the original dataset. This benchmark contains 23 tasks, each with 1000 samples from 10 different classes. **Rotated MNIST** is another variant of MNIST, where each task applies a fixed random image rotation (between 0 and 180 degrees) to the original dataset. This benchmark contains 23 tasks, each with 1000 samples from 10 different classes. **Split CIFAR** is a variant of the CIFAR-100 dataset [68, 156], where each task contains the data pertaining 5 random classes (without replacement) out of the total 100 classes. This benchmark contains 20 tasks, each with 250 samples per each of the 5 classes. **Split miniImageNet** is a variant of the ImageNet dataset [117, 145], containing a subset of images and classes from the original dataset. This benchmark contains 20 tasks, each with 250 samples per each of the 5 classes.

For all datasets, the first 3 tasks are used for hyper-parameter optimization (grids available in Appendix section 6.C). The learners can perform multiple epochs on these three initial tasks, that are later discarded for evaluation.

6.4.2 Baselines

We compare our proposed model HAL to the following baselines. **Finetune** is a single model trained on the stream of data, without any regularization or episodic memory. **ICARL** [110] uses nearest-mean-of-exemplars rule for classification and avoids catastrophic forgetting by regularizing over the feature representations of previous tasks using knowledge distillation loss [57]. **EWC** [67] is a continual learning method that limits changes to parameters critical to past tasks, as measured by the Fisher information matrix. **VCL** [98] is a continual learning method that uses online variational inference for approximating the posterior distribution which is then used to regularize the model. **AGEM** [30] is a continual learning method improving on [87], which uses an episodic memory of parameter gradients to limit forgetting. **MER** [111] is a continual learning method that combines episodic memories with meta-learning to limit forgetting. **ER-Ring** [31] is a continual learning method that uses a ring buffer as episodic memory. **MIR** [4] is a continual learning method based on experience replay that selects a minibatch from the episodic memory that incurrs maximum change in loss. **Multitask** is an oracle baseline that has access to all data to optimize equation 6.1, useful to estimate an upper bound on the obtainable Accuracy (equation 6.2). **Clone-and-finetune** is an oracle baseline training one independent model per task, where the model for task t' is initialized by cloning the parameters of the model for task $t' - 1$.

All baselines use the same neural network architectures: a perceptron with two hidden layers of 256 ReLU neurons in the MNIST experiments, and a ResNet18, with three times less feature maps across all layers, similar to Lopez-Paz and Ranzato [87], in CIFAR and ImageNet experiments. The task identifiers are used to select the output head in the CIFAR and ImageNet experiments, while ignored in the MNIST experiments. Batch size is set to 10 for both the stream of data and episodic memories, across experiments and models. The size of episodic memories is set

between 1 and 5 examples per class per task. The results of VCL are complied by running the official implementation[1], that only works for fully-connected networks, in our continual-learning setup. All the other baselines use our unified code base which is available at https://bit.ly/2mw8bsE. All experiments are averaged over five runs using different random seeds where each seed corresponds to a different dataset ordering among tasks.

6.4.3 Results

Table 6.1 summarizes the main results of our experiments when episodic memory of only one example per class per task is used. First, our proposed HAL is the method achieving maximum Accuracy (equation 6.2) and minimal Forgetting (equation 6.3) at all benchmarks. This does not include oracle baselines *Multitask* (which has access to all data simultaneously) and *Clone-and-finetune* (which trains a separate model per task). Second, the relative gains from the second-best method *ER-Ring* to HAL are significant, confirming that the anchoring objective (equation 6.5) allows experience-replay methods to generalize better from the same amount of episodic memory.

Third, regularization based-approaches, such as EWC [67] and VCL [98], suffer under the single epoch setup. As noted by Chaudhry et al. [30], EWC requires multiple passes over the samples of each task to perform well. The poor performance of VCL is attributed to noisy posterior estimation in the single pass setup. Note that approaches making use of memory (MER, ER, MIR and HAL) work significantly better in this setup.

Fourth, ICARL [110], another method making use of episodic memory, performs poorly in our setup. From the table 6.1, it can be argued that direct training on a very small episodic memory, as done in experience replay, allows the methods

[1] https://github.com/nvcuong/variational-continual-learning

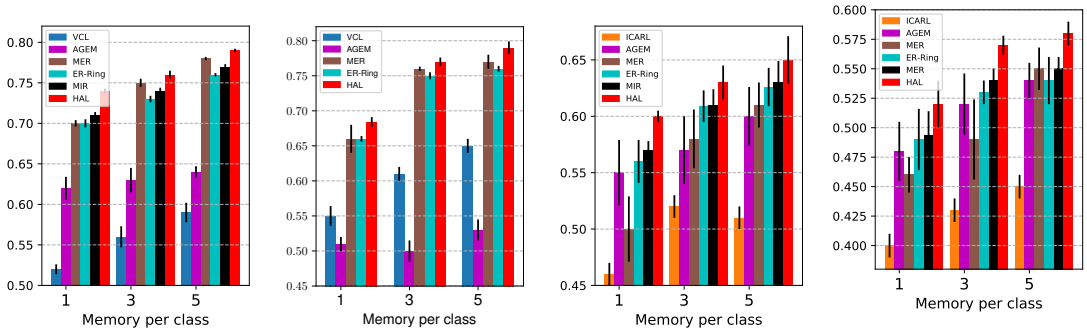
Table 6.1: Accuracy (equation 6.2) and Forgetting (equation 6.3) results of continual learning experiments. Averages and standard deviations are computed over five runs using different random seeds. When used, episodic memories contain up to one example per class per task. The last two rows are oracle baselines.

Method	Permuted MNIST		Rotated MNIST	
	Accuracy	Forgetting	Accuracy	Forgetting
Finetune	53.5 (±1.46)	0.29 (±0.01)	41.9 (±1.37)	0.50 (±0.01)
EWC [67]	63.1 (±1.40)	0.18 (±0.01)	44.1 (±0.99)	0.47 (±0.01)
VCL [98]	51.8 (±1.54)	0.44 (±0.01)	48.2 (±0.99)	0.50 (±0.01)
VCL-Random [98]	52.3 (±0.66)	0.43 (±0.01)	54.4 (±1.44)	0.44 (±0.01)
AGEM [30]	62.1 (±1.39)	0.21 (±0.01)	50.9 (±0.92)	0.40 (±0.01)
MER [111]	69.9 (±0.40)	0.14 (±0.01)	66.0 (±2.04)	0.23 (±0.01)
ER-Ring [31]	70.2 (±0.56)	0.12 (±0.01)	65.9 (±0.41)	0.24 (±0.01)
MIR [4]	71.1 (±0.41)	0.11 (±0.01)	-	-
HAL (ours)	**73.6** (±0.31)	**0.09** (±0.01)	**68.4** (±0.72)	**0.21** (±0.01)
Clone-and-finetune	81.4 (±0.35)	0.0	87.5 (±0.11)	0.0
Multitask	83.0	0.0	83.3	0.0

Method	Split CIFAR		Split miniImageNet	
	Accuracy	Forgetting	Accuracy	Forgetting
Finetune	42.9 (±2.07)	0.25 (±0.03)	34.7 (±2.69)	0.26 (±0.03)
EWC [67]	42.4 (±3.02)	0.26 (±0.02)	37.7 (±3.29)	0.21 (±0.03)
ICARL [110]	46.4 (±1.21)	0.16 (±0.01)	-	-
AGEM [30]	54.9 (±2.92)	0.14 (±0.03)	48.2 (±2.49)	0.13 (±0.02)
MER [111]	49.7 (±2.97)	0.19 (±0.03)	45.5 (±1.49)	0.15 (±0.01)
ER-Ring [31]	56.2 (±1.93)	0.13 (±0.01)	49.0 (±2.61)	0.12 (±0.02)
MIR [4]	57.1 (±1.81)	0.12 (±0.01)	49.3 (±2.15)	0.12 (±0.01)
HAL (ours)	**60.4** (±0.54)	**0.10** (±0.01)	**51.6** (±2.02)	**0.10** (±0.01)
Clone-and-finetune	60.3 (±0.55)	0.0	50.3 (±1.00)	0.0
Multitask	68.3	0.0	63.5	0.0

to generalize better compared to when the same memory is used indirectly in the knowledge distillation loss [57] as done in ICARL.

Figure 6.1 shows the Accuracy (equation 6.2) of methods employing episodic memory when the size of memory is increased. We use 1 to 5 examples per class per task, resulting in a total memory size from 200 to 1000 for MNIST experiments, and



(a) Permuted MNIST (b) Rotated MNIST (c) Split CIFAR (d) Split miniImageNet

Figure 6.1: Accuracy (equation 6.2) results for different episodic memory sizes.

from 85 to 425 for CIFAR and ImageNet experiments. The corresponding numbers for Forgetting are given in Appendix section 6.B. HAL consistently improves on *ER-Ring* and other baselines.

Figure 6.4 in Appendix section 6.B provides the training time of the continual learning baselines on MNIST benchmarks. Although HAL adds an overhead on top of experience replay baseline, it is significantly faster than MER —another approach that makes use of nested optimization to reduce forgetting. However, HAL requires extra memory to store task anchors that, as we will show next, are more effective than additional data samples one can store for experience replay. Overall, we conclude that HAL provides the best trade-off in terms of efficiency and performance.

6.4.4 Ablation Study

We now turn our attention towards two questions; (1) whether for the same episodic memory size in bytes HAL improves over the experience replay baseline, (2) whether fine-tuning on replay buffer is a good approximation of forgetting when the learner is updated on future tasks.

To answer the first question, let $|\mathcal{M}|$ be the total size of episodic memory for

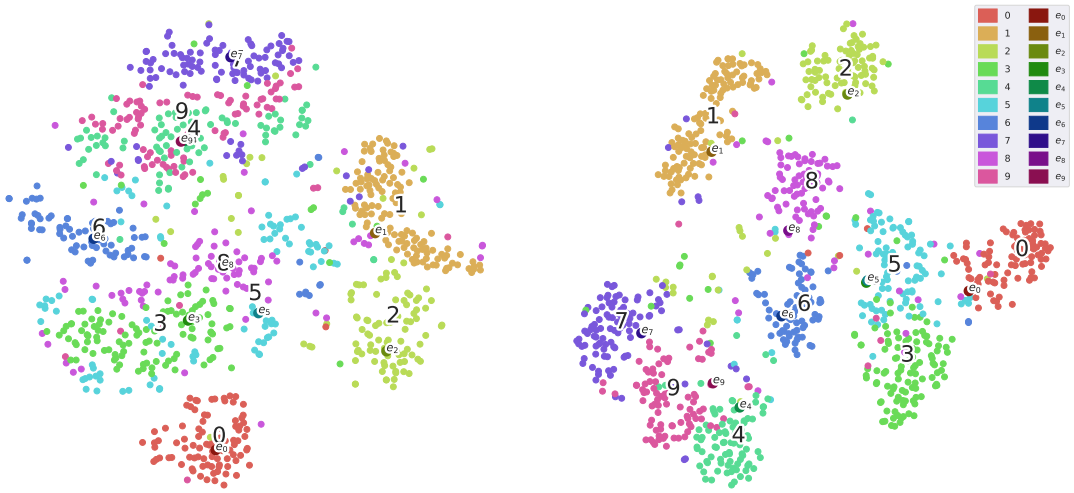


Image Space Feature Space

Figure 6.2: t-SNE visualization of images and anchors (HAL) in the image space (*left*) and the feature space (*right*) on Permuted MNIST benchmark for a single task. Anchor points are exaggerated in size for the purpose of better visualization. The left plot shows that anchor points lie with in the data cluster of a class, whereas the right plot shows that, in the feature space, anchor points lie close to the edge of the cluster of a class or near decision boundaries.

all tasks when one example per class per task is stored in the replay buffer. We then run experience replay with double the size of episodic memory (*i.e.*) storing two examples per class per task instead of one. The episodic memory size in HAL, on the other hand, is kept at $|\mathcal{M}|$. This effectively makes the size of memory in bytes taken by experience replay and that of HAL equal as latter requires extra memory to store anchors. Table 6.2 summarizes the results of this study. For the same memory size in bytes, HAL performs better than experience replay when *additional real data samples are stored in the episodic memory.* It is surprising that the anchors learned by HAL, initialized from random noise and learned using gradient-based optimization, perform better compared to randomly sampled real data. To understand this, in figure 6.2 we visualize HAL's anchors along with the task data in image and feature space on Permuted MNIST benchmark. From the left of the figure, it can be seen that HAL anchors lie with in the data cluster of a class in the image space suggesting that mean embedding loss in equation 6.9

Table 6.2: Comparison of HAL with experience replay. ER-Ring and HAL use one example per class per task in the episodic memory, whereas ER-Ring-2$|\mathcal{M}|$ uses two examples per class per task in the memory. Averages and standard deviations are computed over five runs using different random seeds.

Method	**Permuted MNIST**		**Split CIFAR**	
	Accuracy	Forgetting	Accuracy	Forgetting
ER-Ring-$|\mathcal{M}|$	70.2 $\pm$(0.56)	0.12 ($\pm$0.01)	56.2 ($\pm$1.93)	0.13 ($\pm$0.01)
ER-Ring-2$|\mathcal{M}|$	71.9 ($\pm$0.31)	0.11 ($\pm$0.01)	58.6 ($\pm$2.68)	0.12 ($\pm$0.01)
HAL-$|\mathcal{M}|$	**73.6** ($\pm$0.31)	**0.09** ($\pm$0.01)	**60.4** ($\pm$0.54)	**0.10** ($\pm$0.01)

Table 6.3: Performance comparison of HAL with Oracle where the learner has access to all the future tasks to exactly quantify forgetting of an anchor. Averages and standard deviations are computed over five runs using different random seeds.

Anchor type	**Permuted MNIST**		**Split CIFAR**	
	Accuracy	Forgetting	Accuracy	Forgetting
HAL	73.6 ($\pm$0.31)	0.09 ($\pm$0.01)	60.4 ($\pm$0.54)	0.10 ($\pm$0.01)
Oracle	73.9 ($\pm$0.41)	0.09 ($\pm$0.01)	61.1 ($\pm$0.94)	0.09 ($\pm$0.01)

effectively regularizes against outliers. More interestingly, figure 6.2 (right column) shows that these anchors lie at or close to the cluster edges in the feature space. In other words, anchor points learned by HAL lie close to the classifier decision boundary. This can explain their effectiveness compared to the real data samples that can lie anywhere in the data cluster in feature space.

Finally, to answer the second part, we assume a non-continual setup where at each step the learner has an oracle access to all future tasks. After training on task t, the learner is fine-tuned on all future tasks and anchor points are subsequently learned by optimizing idealistic equation 6.7. The results are reported in table 6.3. It can be seen from the table that the proposed HAL performs very close to the non-continual oracle baseline. This suggests that HAL's approximation of forgetting when the learner is updated on future tasks by replaying past data is effective in many existing continual learning benchmarks.

6.5 Related work

In continual learning [112], also called lifelong learning [140], a learner addresses a *sequence* of changing tasks without storing the complete datasets of these tasks. This is in contrast to *multitask learning* [27], where the learner assumes simultaneous access to data from all tasks. The main challenge in continual learning is to avoid catastrophic interference [93, 92, 48], that is, the learner forgetting previously acquired knowledge when learning new tasks. The state-of-the art methods in continual learning can be categorized into three classes.

First, *regularization approaches* discourage updating parameters important for past tasks [67, 3, 98, 156]. While efficient in terms of memory and computation, these approaches suffer from brittleness due to feature drift for large number of tasks [141]. Additionally, these approaches are only effective when the learner can perform multiple passes over each task [30], a case deemed unrealistic in this work.

Second, *modular approaches* use different parts of the prediction function for each new task [40, 2, 115, 28, 153, 41]. Modular approaches do not scale to a large number of tasks, as they require searching over combinatorial space of module architectures. Another modular approach [118, 80] adds new parts to the prediction function as new tasks are learned. By construction, modular approaches have zero forgetting, but their memory requirements increase with the number of tasks.

Third, *episodic memory approaches* maintain and revisit a small episodic memory of data from past tasks. In some of these methods [85, 110], examples in the episodic memory are replayed and predictions are kept invariant by means of distillation [57]. In other approaches [87, 30, 6] the episodic memory is used as an optimization constraint that discourages increases in loss at past tasks. More recently, several works [52, 111, 114, 31] have shown that directly optimizing the loss on the episodic memory, also known as experience replay, is cheaper than constraint-based approaches and improves prediction performance. Our contribu-

tion in this chapter has been to improve *experience replay methods* with task anchors learned in hindsight.

There are other definitions of continual learning, such as the one of task-free continual learning. The task-free formulation does not consider the notion of tasks, and instead works on undivided data streams [5, 6]. We have focused on the task-based definition of continual learning and, similar to many recent works [87, 52, 111, 30], assumed that only a *single pass through the data* was possible.

Finally, our gradient-based learning of anchors bears a similarity to [132] and [149]. In Simonyan et al. [132], the authors use gradient ascent on class scores to find saliency maps of a classification model. Contrary to them, our proposed hindsight learning objective optimizes for the forgetting metric, as reducing the said metric is necessary for continual learning. Wang et al. [149] proposes dataset distillation that encodes the entire dataset in a few synthetic points at a given parameter vector by a gradient-based optimization process. Their method requires access to the entire dataset of a task for optimization purposes. We, instead, learn anchors in hindsight from the replay buffer of past tasks *after* training is finished for current task. While Wang et al. [149] aim to replicate the performance of the entire dataset from the synthetic points, we focus on reducing forgetting of an already learned task.

6.6 Conclusion

In this chapter, we introduced a bilevel optimization objective, dubbed anchoring, for continual learning. In our approach, we learned one "anchor point" per class per task, where predictions are requested to remain invariant by the means of nested optimization. These anchors are learned using gradient-based optimization, and represent points that would maximize the forgetting of the current task throughout the entire learning experience. We simulate the forgetting that would happen

during the learning of future tasks *in hindsight*, that is, by taking temporary gradient steps across a small episodic memory of past tasks. We call our approach Hindsight Anchor Learning (HAL). As shown in our experiments, anchoring in hindsight complements and improves the performance of continual learning methods based on experience replay, achieving a new state of the art on four standard continual learning benchmarks.

Appendix

Section 6.A describes the approximate update performed by anchoring objective (equation 6.5 in the main chapter 6). Section 6.B reports more experimental results. Section 6.C lists hyperparameters for all experiments whereas section 6.D quantifies the sensitivity of the proposed algorithm to the hyper-parameter settings. Section 6.E gives pseudo-code for HAL.

6.A Approximate Update by Anchoring Objective

Here we will use a Taylor series expansion to approximate the update performed by anchoring objective (equation 6.5 in the main chapter 6). In particular, we are interested in the regularization part of the anchoring objective that involves nested update. We refer to this gradient as g_{anc}. We follow similar arguments as that of Nichol and Schulman [99].

Proof. Let θ_0 be the parameter vector before the temporary update in the anchoring objective (equation 6.5). Also, let ℓ_{ce} and ℓ_{L2} be the cross-entropy and L2 losses, respectively. We use the following definitions:

$$\overline{g}_0 = \ell'_{ce}(\theta_0). \qquad \text{(gradient of cross-entropy loss at initial point on } \mathcal{B} \cup \mathcal{B}_{\mathcal{M}})$$

$$\overline{H}_0 = \ell''_{ce}(\theta_0). \qquad \text{(Hessian of cross-entropy loss at initial point on } \mathcal{B} \cup \mathcal{B}_{\mathcal{M}})$$

$$\overline{g}_1 = \ell'_{L2}(\theta_0). \qquad \text{(gradient of L2 loss at initial point on anchors)}$$

$$\overline{H}_1 = \ell''_{L2}(\theta_0). \qquad \text{(Hessian of L2 loss at initial point on anchors)}$$

Let $U_0 = \theta_0 - \alpha\overline{g}_0$ be the operator giving a temporary update in the two-step process of (equation 6.5), and let θ_1 be the temporary update itself *(i.e.)* $\theta_1 := U_0$

(note that $\tilde{\theta}$ is used in the main chapter 6 instead of θ_1). The g_{anc} is given by:

$$\begin{aligned} g_{anc} &= \frac{\partial}{\partial \theta_0} \ell_{L2}(U_0), \\ &= U_0' \cdot \ell_{L2}'(\theta_1), \\ &= \left(I - \alpha \overline{H}_0\right) \cdot \ell_{L2}'(\theta_1), \end{aligned} \tag{6.10}$$

where the second step is obtained by using chain rule. Now, if we calculate the first order Taylor series approximation of $\ell_{L2}'(\theta_1)$,

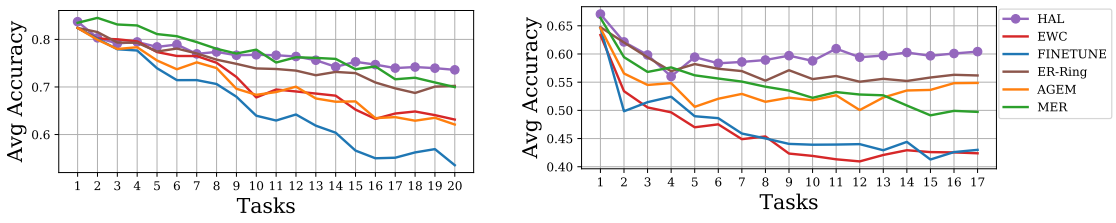
$$\begin{aligned} \ell_{L2}'(\theta_1) &= \ell_{L2}'(\theta_0) + \ell_{L2}''(\theta_0) \cdot (\theta_1 - \theta_0) + O(||\theta_1 - \theta_0||^2), \\ &= \overline{g}_1 + \overline{H}_1 \cdot (\theta_0 - \alpha \overline{g}_0 - \theta_0) + O(\alpha^2), \\ &= \overline{g}_1 - \alpha \overline{H}_1 \cdot \overline{g}_0 + O(\alpha^2), \end{aligned} \tag{6.11}$$

where in the second step we substituted the value of θ_1. By putting equation 6.11 in equation 6.10 and after some simplification we get:

$$g_{anc} = \overline{g}_1 - \alpha(\overline{H}_1 \cdot \overline{g}_0 + \overline{H}_0 \cdot \overline{g}_1) + O(\alpha^2). \tag{6.12}$$

This form is very similar to the second-order MAML gradient formulation, equation 25 in [99]. Further simplification of the inner product terms between Hessian and gradients yields inner product between the gradients $\overline{g_0}$ and $\overline{g_1}$. □

This shows that similar to MAML [42], Reptile [99] and MER [111], anchoring objective, as described in equation 6.5 of the main chapter 6, maximizes the inner product between the gradients. However, unlike the other meta-learning approaches, in anchoring objective, these gradients correspond to different loss



Permuted MNIST Split CIFAR

Figure 6.3: Evolution of Accuracy (equation 6.2) as new tasks are learned. When used, episodic memories contain up to one example per class per task.

Table 6.4: Impact of anchor selection, where we compare a randomly chosen data point as an anchor (Real Data Anchor) with our optimized anchor selection (HAL).

Anchor type	**Split CIFAR**	
	Accuracy	Forgetting
Real Data Anchor	58.0 ($\pm$0.15)	0.12 ($\pm$0.01)
HAL (ours)	60.4 ($\pm$0.54)	0.10 ($\pm$0.01)

functions, cross-entropy and L2 losses on data from current task and episodic memory, and HAL anchors, respectively.

6.B More Results

Figure 6.3 shows a more fine grained analysis of average accuracy as new tasks are learned on Permuted MNIST and Split CIFAR. HAL preserves the performance of a predictor more effectively than other baselines.

Tables 6.5 and 6.6 show the Accuracy and Forgetting of methods employing episodic memory when the size of memory is increased. We use 3 to 5 examples per class per task, resulting in a total memory size from 600 to 1000 for MNIST experiments, and from 255 to 425 for CIFAR and ImageNet experiments.

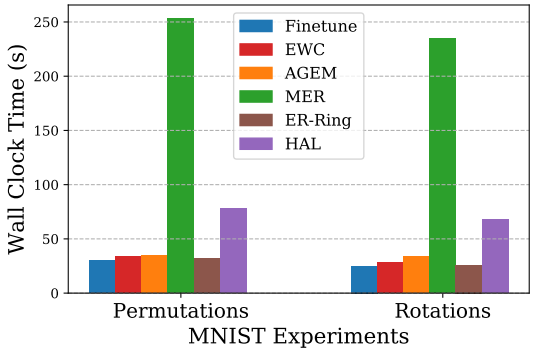


Figure 6.4: Training time (s) of MNIST experiments for the entire continual learning experience. MER and HAL both use meta-learning objectives to reduce forgetting.

Table 6.5: Accuracy (equation 6.2) results for large (3 to 5 examples per class per task) episodic memory sizes. Here we only compare methods that use an episodic memory. Metrics are averaged over five runs using different random seeds.

Method	Permuted MNIST		Rotated MNIST	
	$\|\mathcal{M}\| = 600$	$\|\mathcal{M}\| = 1000$	$\|\mathcal{M}\| = 600$	$\|\mathcal{M}\| = 1000$
VCL-Random	55.8 ($\pm$1.29)	58.5 ($\pm$1.21)	61.2 ($\pm$0.12)	64.4 ($\pm$0.16)
AGEM	63.2 ($\pm$1.47)	64.1 ($\pm$0.74)	49.9 ($\pm$1.49)	53.0 ($\pm$1.52)
MER	74.9 ($\pm$0.49)	78.3 ($\pm$0.19)	76.5 ($\pm$0.30)	77.3 ($\pm$1.13)
ER-Ring	73.5 ($\pm$0.43)	75.8 ($\pm$0.24)	74.7 ($\pm$0.56)	76.5 ($\pm$0.48)
HAL (ours)	**76.2** ($\pm$0.52)	**78.4** ($\pm$0.27)	**77.0** ($\pm$0.66)	**78.7** ($\pm$0.97)

Method	Split CIFAR		Split miniImageNet	
	$\|\mathcal{M}\| = 255$	$\|\mathcal{M}\| = 425$	$\|\mathcal{M}\| = 255$	$\|\mathcal{M}\| = 425$
ICARL	51.7 ($\pm$1.41)	51.2 ($\pm$1.32)	-	-
AGEM	56.9 ($\pm$3.45)	59.9 ($\pm$2.64)	51.6 ($\pm$2.69)	54.3 ($\pm$1.56)
MER	57.7 ($\pm$2.59)	60.6 ($\pm$2.09)	49.4 ($\pm$3.43)	54.8 ($\pm$1.79)
ER-Ring	60.9 ($\pm$1.44)	62.6 ($\pm$1.77)	53.5 ($\pm$1.42)	54.2 ($\pm$3.23)
HAL (ours)	**62.9** ($\pm$1.49)	**64.4** ($\pm$2.15)	**56.5** ($\pm$0.87)	**57.2** ($\pm$1.54)

6.C Hyperparameters

Table 6.7 lists the best hyperparameters for each experiment.

Table 6.6: Forgetting (equation 6.3) results for large (3 to 5 examples per class per task) episodic memory sizes. Here we only compare methods that use an episodic memory. Averages and standard deviations are computed over five runs using different random seeds.

METHOD	PERMUTED MNIST		ROTATED MNIST	
	$\|\mathcal{M}\| = 600$	$\|\mathcal{M}\| = 1000$	$\|\mathcal{M}\| = 600$	$\|\mathcal{M}\| = 1000$
VCL-RANDOM	0.39 ($\pm$0.01)	0.36 ($\pm$0.01)	0.37 ($\pm$0.01)	0.33 ($\pm$0.01)
AGEM	0.20 ($\pm$0.01)	0.19 ($\pm$0.01)	0.41 ($\pm$0.01)	0.38 ($\pm$0.01)
MER	0.14 ($\pm$0.01)	0.09 ($\pm$0.01)	**0.12** ($\pm$0.01)	**0.11** ($\pm$0.01)
ER-RING	0.09 ($\pm$0.01)	0.07 ($\pm$0.01)	0.15 ($\pm$0.01)	0.13 ($\pm$0.01)
HAL (OURS)	**0.07** ($\pm$0.01)	**0.05** ($\pm$0.01)	**0.12** ($\pm$0.01)	**0.11** ($\pm$0.01)

METHOD	SPLIT CIFAR		SPLIT MINIIMAGENET	
	$\|\mathcal{M}\| = 255$	$\|\mathcal{M}\| = 425$	$\|\mathcal{M}\| = 255$	$\|\mathcal{M}\| = 425$
ICARL	0.13 ($\pm$0.02)	0.13 ($\pm$0.02)	-	-
AGEM	0.13 ($\pm$0.03)	0.10 ($\pm$0.02)	0.10 ($\pm$0.02)	0.08 ($\pm$0.01)
MER	0.11 ($\pm$0.01)	0.09 ($\pm$0.02)	0.12 ($\pm$0.02)	0.07 ($\pm$0.01)
ER-RING	0.09 ($\pm$0.01)	**0.06** ($\pm$0.01)	0.07 ($\pm$0.02)	0.08 ($\pm$0.02)
HAL (OURS)	**0.08** ($\pm$0.01)	**0.06** ($\pm$0.01)	**0.06** ($\pm$0.01)	**0.06** ($\pm$0.01)

6.D Hyperparameter Sensitivity

In table 6.8, we report the performance of HAL against a range of hyperparameters. For a given hyperparameter in the table, all the other hyperparameters are set to their optimal values found in section 6.C of the appendix. HAL is not sensitive to the choice of hyperparameters.

6.E HAL Algorithm

Algorithm 9 provides pseudo-code for HAL.

Algorithm 9 Training of HAL on sequential data $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$, with total replay buffer size 'mem_sz', learning rate 'α', regularization strength 'λ', mean embedding decay 'β', mean embedding strength 'η'.

procedure HAL($\mathcal{D}$, mem_sz, α, λ, β)
  $\mathcal{M} \leftarrow \{\} *$ mem_sz
  $\{e_1, \cdots, e_T\} \leftarrow \{\}$
  for $t \in \{1, \cdots, T\}$ **do**
    $\phi_t \leftarrow \vec{0}$
    for $\mathcal{B} \sim \mathcal{D}_t$ **do** ▷ Sample a batch from current task
      $\mathcal{B}_\mathcal{M} \sim \mathcal{M}$ ▷ Sample a batch from episodic memory
      $\tilde{\theta} \leftarrow \theta - \alpha \cdot \nabla_\theta \, \ell(\mathcal{B} \cup \mathcal{B}_\mathcal{M})$ ▷ Temporary parameter update
      $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \left(\ell(\mathcal{B} \cup \mathcal{B}_\mathcal{M}) + \lambda \cdot \sum_{t'<t} \left(f_\theta(e_{t'}, t') - f_{\tilde{\theta}}(e_{t'}, t') \right)^2 \right)$ ▷ Anchoring objective (equation 6.5)
      $\phi_t \leftarrow \beta \cdot \phi_t + (1-\beta) \cdot \phi(\mathcal{B})$ ▷ Running average of mean embedding
      $\mathcal{M} \leftarrow$ UpdateMemory($\mathcal{M}, \mathcal{B}$) ▷ Add samples to a ring buffer
    end for
    $e_t, \theta \leftarrow$ GetAnchors($\mathcal{M}, \theta, \phi_t, \eta$) ▷ Get anchors for current task
  end for
  return $\theta, \mathcal{M}$
end procedure

procedure GetAnchors($\mathcal{M}, \theta_t, \phi_t, \gamma$)
  $\theta \leftarrow \theta_t$
  for $\mathcal{B}_\mathcal{M} \sim \mathcal{M}$ **do**
    $\theta \leftarrow \theta - \alpha \cdot \nabla_\theta \ell(\mathcal{B}_\mathcal{M})$ ▷ Finetune θ_t by taking SGD steps on the episodic memory
  end for
  $\theta_\mathcal{M} \leftarrow \theta$ ▷ Store the updated parameter
  $e_t \leftarrow$ rand() ▷ Initialize the task anchors
  for $1, \cdots, k$ **do**
    $e_t \leftarrow e_t + \alpha \cdot \nabla_{e_t} \left(\ell(f_{\theta_\mathcal{M}}(e_t, t), y_t) - \ell(f_{\theta_t}(e_t, t), y_t) - \gamma(\phi(e_t) - \phi_t)^2 \right)$ ▷ Maximize forgetting (equation 6.9)
  end for
  return e_t, θ_t
end procedure

Table 6.7: Hyperparameters for all the baselines.

Method	Dataset	Hyperparameters
Multitask	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
Finetune	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
EWC	Permuted MNIST	learning rate (0.1), regularization (10)
	Rotated MNIST	learning rate (0.1), regularization (10)
	Split CIFAR-100	learning rate (0.03), regularization (10)
	Split miniImageNet	learning rate (0.03), regularization (10)
A-GEM	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
MER	Permuted MNIST	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (10)
	Rotated MNIST	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (10)
	Split CIFAR-100	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (5)
	Split miniImageNet	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (5)
ER-Ring	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
HAL	Permuted MNIST	learning rate (0.1), λ (0.1), γ (0.1), β (0.5), gradient steps on anchors (100)
	Rotated MNIST	learning rate (0.1), λ (0.1), γ (0.1), β (0.5), gradient steps on anchors (100)
	Split CIFAR-100	learning rate (0.03), λ (1.0), γ (0.1), β (0.5), gradient steps on anchors (100)
	Split miniImageNet	learning rate (0.03), λ (0.3), γ (0.1), β (0.5), gradient steps on anchors (100)

Table 6.8: Average Accuracy of HAL on different values of hyperparameters. For a given hyperparameter in the table, all the other hyperparameters are set to their optimal values found in section 6.C of the appendix.

Dataset	λ	Acc	γ	Acc	β	Acc
Permuted MNIST	0.01	72.8 $\pm$(0.52)	0.01	73.1 $\pm$(0.20)	0.1	72.5 $\pm$(0.95)
	0.1	73.6 $\pm$(0.31)	0.1	73.6 $\pm$(0.31)	0.5	73.6 $\pm$(0.31)
	1.0	73.2 $\pm$(0.85)	1.0	73.4 $\pm$(0.41)	0.9	72.9 $\pm$(0.39)
Split CIFAR100	0.01	58.5 $\pm$(1.25)	0.01	59.8 $\pm$(0.65)	0.1	58.7 $\pm$(1.17)
	0.1	59.2 $\pm$(0.91)	0.1	60.4 $\pm$(0.54)	0.5	60.4 $\pm$(0.54)
	1.0	60.4 $\pm$(0.54)	1.0	60.2 $\pm$(1.21)	0.9	59.6 $\pm$(1.05)

Chapter 7

Continual Learning in Low-rank Orthogonal Subspaces

Abstract

In continual learning (CL), a learner is faced with a sequence of tasks, arriving one after the other, and the goal is to remember all the tasks once the continual learning experience is finished. The prior art in CL uses episodic memory, parameter regularization or extensible network structures to reduce interference among tasks, but in the end, all the approaches learn different tasks in a joint vector space. We believe this invariably leads to interference among different tasks. We propose to learn tasks in different (low-rank) vector subspaces that are kept orthogonal to each other in order to minimize interference. Further, to keep the gradients of different tasks coming from these subspaces orthogonal to each other, we learn isometric mappings by posing network training as an optimization problem over the Stiefel manifold. We report strong results over experience-replay baseline with and without memory on standard classification benchmarks in continual learning.

7.1 Introduction

In continual learning, a learner experiences a sequence of tasks with the objective to remember all or most of the observed tasks to speed up transfer of knowledge to future tasks. Learning from a diverse sequence of tasks is useful as it allows for the deployment of machine learning models that can quickly adapt to changes in the environment by leveraging past experiences. Contrary to the standard supervised learning setting, where only a single task is available, and where the learner can make several passes over the dataset of the task, the sequential arrival of multiple tasks poses unique challenges for continual learning. The chief one among which is catastrophic forgetting [93], whereby the global update of model parameters on the present task interferes with the learned representations of past tasks. This results in the model forgetting the previously acquired knowledge.

In neural networks, to reduce the deterioration of accumulated knowledge, existing approaches modify the network training broadly in three different ways. First, *regularization-based* approaches [67, 156, 3, 29, 98] reduce the drift in network parameters that were important for solving previous tasks. Second, *modular* approaches [118, 80], add network components as new tasks arrive. These approaches rely on the knowledge of correct module selection at test time. Third, and perhaps the strongest, *memory-based* approaches [87, 52, 60, 111], maintain a small replay buffer, called episodic memory, and mitigate catastrophic forgetting by replaying the data in the buffer along with the new task data. One common feature among all the three categories is that, in the end, all the tasks are learned in the same vector space where a vector space is associated with the output of a hidden layer of the network. We believe this restriction invariably leads to forgetting of past tasks.

In this chapter, we propose to learn different tasks in different vector subspaces. We require these subspaces to be orthogonal to each other in order to prevent the learning of a task from interfering catastrophically with previous ones. More

specifically, for a point in the vector space in $\mathbb{R}^m$, typically the second last layer of the network, we project each task to a low-dimensional subspace by a task-specific projection matrix $P \in \mathbb{R}^{m \times m}$, whose rank is r, where $r \ll m$. The projection matrices are generated offline such that for different tasks they are mutually orthogonal. This simple projection in the second last layer reduces the forgetting considerably in the shallower networks – the average accuracy increases by up to 13% and forgetting drops by up to 66% compared to the strongest experience replay baseline [31] in a three-layer network. However, in deeper networks, the backpropagation of gradients from the different projections of the second last layer do not remain orthogonal to each other in the earlier layers resulting in interference in those layers. To reduce the interference, we use the fact that a gradient on an earlier layer is a transformed version of the gradient received at the projected layer – where the transformation is linear and consists of the product of the weight matrix and the diagonal Jacobian matrix of the non-linearity of the layers in between. Reducing interference then requires this transformation to be an inner-product preserving transformation, such that, if two vectors are orthogonal at the projected layer, they remain close to orthogonal after the transformation. This is equivalent to learning orthonormal weight matrices – a well-studied problem of learning on Stiefel manifolds [1, 23]. Our approach, dubbed ORTHOG-SUBSPACE, generates two projected orthogonal vectors (gradients) – one for the current task and another for one of the previous tasks whose data is stored in a tiny replay buffer – and updates the network weights such that the weights remain on a Stiefel manifold. We visually describe our approach in figure 7.1. For the same amount of episodic memory, ORTHOG-SUBSPACE, improves upon the strong experience replay baseline by 8% in average accuracy and 50% in forgetting on deeper networks.

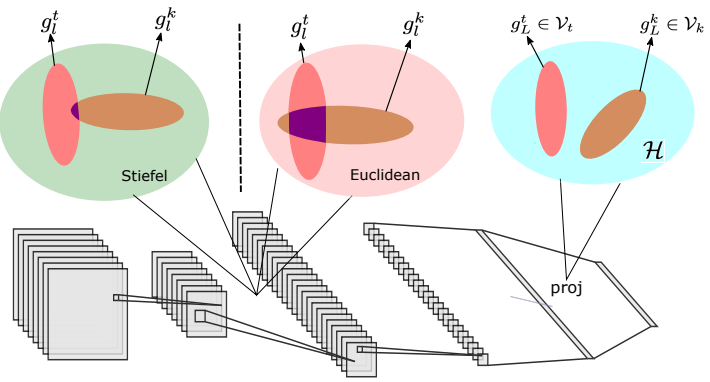


Figure 7.1: ORTHOG-SUBSPACE. Each blob, with the three ellipses, represents a vector space and its subspaces at a certain layer. The projection operator in the layer L keeps the subspaces orthogonal (no overlap). The overlap in the intermediate layers is minimized when the weight matrices are learned on the Stiefel manifold.

7.2 Background

In this section, we describe the continual learning setup and preliminaries for our approach.

7.2.1 Continual Learning Setup

We assume a continual learner experiencing a stream of data triplets (x_i, y_i, t_i) containing an input x_i, a target y_i, and a task identifier $t_i \in \mathcal{T} = \{1, \ldots, T\}$. Each input-target pair $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}_{t_i}$ is an identical and independently distributed example drawn from some unknown distribution $P_{t_i}(X, Y)$, representing the t_i-th learning task. We assume that the tasks are experienced in order, $t_i \leq t_j$ for all $i \leq j$, and the learner cannot store any but a few samples from P_{t_i} in a tiny replay buffer $\mathcal{M}_i$. Under this setup, our goal is to estimate a predictor $f = (w \circ \Phi) : \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{Y}$, composed of a feature extractor $\Phi_\Theta : \mathcal{X} \rightarrow \mathcal{H}$, which is an L-layer feed-forward neural network parameterized by $\Theta = \{W_l\}_{l=1}^L$, and a classifier $w_\theta : \mathcal{H} \rightarrow \mathcal{Y}$, that

minimizes the multi-task error

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{(x,y)\sim P_t}\left[\,\ell(f(x,t),y)\,\right], \tag{7.1}$$

where $\mathcal{H}\in\mathbb{R}^m$ is an inner product space, $\mathcal{Y}=\cup_{t\in\mathcal{T}}\mathcal{Y}_t$, and $\ell:\mathcal{Y}\times\mathcal{Y}\to\mathbb{R}_+$ is a loss function.

To further comply with the strict sequential setting, similar to prior work [87, 111], we consider streams of data that are *experienced only once*. We only focus on classification tasks where either input or output distribution changes over time. We assume that a task descriptor, identifying the correct classification head, is given at both train and test times.

Metrics

Once the continual learning experience is finished, we measure two statistics to evaluate the quality of the algorithms: *average accuracy*, and *average maximum forgetting*. First, the average accuracy is defined as

$$\text{Accuracy}=\frac{1}{T}\sum_{j=1}^{T}a_{T,j}, \tag{7.2}$$

where $a_{i,j}$ denotes the test accuracy on task j after the model has finished experiencing task i. Second, the average maximum forgetting is defined as

$$\text{Forgetting}=\frac{1}{T-1}\sum_{j=1}^{T-1}\max_{l\in\{1,\ldots,T-1\}}(a_{l,j}-a_{T,j}), \tag{7.3}$$

that is, the decrease in performance at each of the tasks between their peak accuracy and their accuracy after the continual learning experience is finished.

7.2.2 Preliminaries

Let the inner product in $\mathcal{H}$ be denoted by $\langle \cdot, \cdot \rangle$, and v be an element of $\mathcal{H}$. A matrix $O \in \mathbb{R}^{m \times r}$, where $r \ll m$, parameterizes an $m \times m$ dimensional orthogonal projection matrix P, given by $P = O(O^\top O)^{-1} O^\top$, where rank$(P) = r$. A vector $u = Pv$, will be the projection of v in a subspace $\mathcal{U} \subset \mathcal{H}$ with dim$(\mathcal{U}) = r$. Further, if the columns of O are assumed to be orthonormal, then the projection matrix is simplified to $P = OO^\top$.

Definition 7.2.1 (Orthogonal Subspace)**.** Subspaces $\mathcal{U}$ and $\mathcal{W}$ of a vector space $\mathcal{H}$ are orthogonal if

$$\langle u, w \rangle = 0, \quad \forall u \in \mathcal{U}, w \in \mathcal{W}.$$

Definition 7.2.2 (Isometry)**.** A linear transformation $T : \mathcal{V} \to \mathcal{V}$ is called an isometry if it is distance preserving *i.e.*

$$\|T(v) - T(w)\| = \|v - w\|, \quad \forall v, w \in \mathcal{V}.$$

A linear transformation that preserves distance, must preserve angles and vice-versa. We record this in the following theorem.

Theorem 7.2.1. T is an isometry iff it preserves inner products.

The proof is fairly standard and given in Appendix section 7.A for completeness.

Corollary 7.2.1.1. If T_1 and T_2 are two isometries then their composition $T_1 \circ T_2$ is also an isometry.

An *orthogonal matrix* preserves inner products and therefore acts as an isometry of Euclidean space. Enforcing orthogonality[1] during network training corresponds to solving the following constrained optimization problem:

$$\begin{aligned} &\min_{\theta, \Theta=\{W_l, b_l\}_{l=1}^{L}} \ell(f(x,t),y), \\ &\text{s.t.} \quad W_l^\top W_l = \mathbf{I}, \quad \forall l \in \{1, \cdots, L\}, \end{aligned} \tag{7.4}$$

where $\mathbf{I}$ is an identity matrix of appropriate dimensions. The solution set of the above problem is a valid Riemannian manifold when the inner product is defined. It is called the Stiefel manifold, defined as $\bar{\mathcal{M}}_l = \{W_l \in \mathbb{R}^{n_l \times n_{l-1}} | W_l^\top W_l = \mathbf{I}\}$, where n_l is the number of neurons in layer l, and it is assumed that $n_l \geq n_{l-1}$. For most of the neural network architectures this assumption holds. For a convolutional layer $W_l \in \mathbb{R}^{c_{out} \times c_{in} \times h \times w}$, we reshape the weight to $W_l \in \mathbb{R}^{c_{out} \times (c_{in} \cdot h \cdot w)}$.

The optimization of a differentiable cost function over a Stiefel manifold has been extensively studied in literature [1, 23]. Here, we briefly summarize the two main steps of the optimization process and refer the reader to Absil et al. [1] for further details. For a given point W_l on the Stiefel manifold, let $\mathcal{T}_{W_l}$ represent the tangent space at that point. Further, let g_l, a matrix, be the gradient of the loss function with respect to W_l. The first step of optimization projects the g_l to $\mathcal{T}_{W_l}$ using a closed form $Proj_{\mathcal{T}_{W_l}}(g_l) = AW_l$, where '$A$' is a skew-symmetric matrix given by (see Appendix section 7.B for the derivation):

$$A = g_l W_l^\top - W_l g_l^\top. \tag{7.5}$$

Once the gradient projection in the tangent space is found, the second step is to generate a descent curve of the loss function in the manifold. The Cayley transform

[1]Note, an orthogonal matrix is always square. However, the matrices we consider can be non-square. In this work, the orthogonal matrix is used in the sense of $W^\top W = \mathbf{I}$.

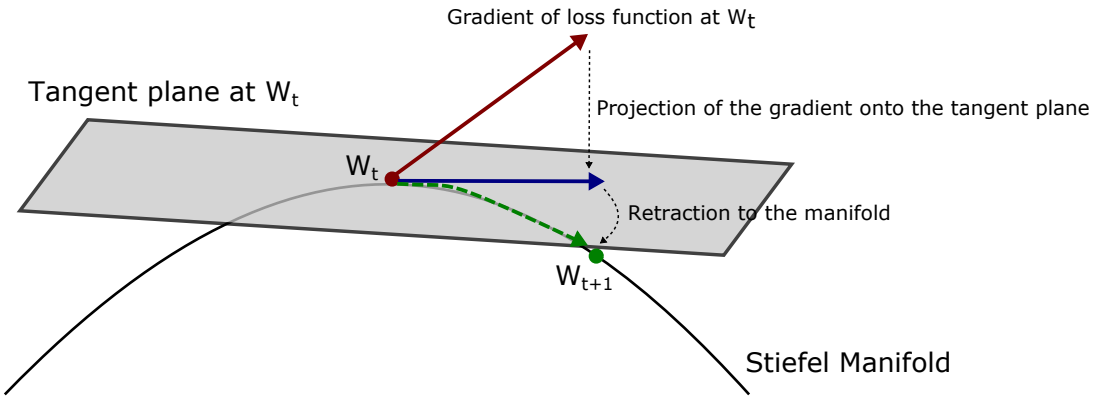


Figure 7.2: Gradient computed at a given point (W_t) in the manifold is first projected to the tangent plane. There exists a closed form for this step. This projected gradient is then retracted to a point in the manifold giving the final update W_{t+1}.

defines one such curve using a parameter $\tau \geq 0$, specifying the length of the curve, and a skew-symmetric matrix U [100]:

$$Y(\tau) = \Big(I + \frac{\tau}{2}U\Big)^{-1}\Big(I - \frac{\tau}{2}U\Big)W_l, \tag{7.6}$$

It can be seen that the curve stays on the Stiefel manifold *i.e.* $Y(\tau)^\top Y(\tau) = \mathbf{I}$ and $Y(0) = W_l$, and that its tangent vector at $\tau = 0$ is $Y'(0) = -UW_l$. By setting $U = A = g_l W_l^\top - W_l g_l^\top$, the curve will be a descent curve for the loss function. Li et al. [84] showed that one can bypass the expensive matrix inversion in equation 7.6 by following the fixed-point iteration of the Cayley transform,

$$Y(\tau) = W_l - \frac{\tau}{2}A(W_l + Y(\tau)). \tag{7.7}$$

Li et al. [84] further showed that under some mild continuity assumptions equation 7.7 converges to the closed-form equation 7.6 faster than other approximation algorithms. The overall optimization on Stiefel manifold is shown in figure 7.2.

7.3 Continual Learning in Orthogonal Subspaces

We now describe our continual learning approach. Consider a feature extractor in the form of a feed-forward neural network consisting of L hidden layers, that takes an input $x \in \mathbb{R}^d$ and passes it through the following recursions: $h_l = \sigma(W_l h_{l-1} + b_l)$, where $\sigma(\cdot)$ is a non-linearity, $h_0 = x$, and $h_L = \phi \in \mathbb{R}^m$. The network is followed by an application-specific head (*e.g.*) a classifier in case of a classification task. The network can be thought of as a mapping, $\Phi : \mathcal{X} \to \mathcal{H}$, from one vector space ($\mathcal{X} \in \mathbb{R}^d$) to another ($\mathcal{H} \in \mathbb{R}^m$). When the network is trained for more than one tasks, a shared vector space $\mathcal{H}$ can be learned if the model has a simultaneous access to all the tasks. In continual learning, on the other hand, when tasks arrive in a sequence, learning a new task can interfere in the space where a previous task was learned. This can result in the catastrophic forgetting of the previous task if the new task is different from the previous one(s). In this work, we propose to learn tasks in *orthogonal subspaces* such that learning of a new task has minimal interference with already learned tasks.

We assume that the network is sufficiently parameterized, which often is the case with deep networks, so that all the tasks can be learned in independent subspaces. We define a family of sets $\mathcal{V}$ that partitions $\mathcal{H}$, such that, $a)$ $\mathcal{V}$ does not contain the empty set ($\emptyset \notin \mathcal{V}$), $b)$ the union of sets in $\mathcal{V}$ is equal to $\mathcal{H}$ ($\cup_{\mathcal{V}_t \in \mathcal{V}} \mathcal{V}_t = \mathcal{H}$), and $c)$ the intersection of any two distinct sets in $\mathcal{V}$ is empty ($(\forall \mathcal{V}_i, \mathcal{V}_j \in \mathcal{V})\ i \neq j \implies \mathcal{V}_i \cap \mathcal{V}_j = \emptyset$)). A set $\mathcal{V}_t \in \mathcal{V}$ defines a subspace for task t. We obtain such a subspace by projecting the feature map $\phi = h_L \in \mathbb{R}^m$ into an r-dimensional space, where $r \ll m$, via a projection matrix $P_t \in \mathbb{R}^{m \times m}$ of rank r, *i.e.* we obtain the features for

task t by $\phi_t = P_t h_L$, while ensuring:

$$\begin{aligned} P_t^\top P_t &= \mathbf{I}, \\ P_t^\top P_k &= \mathbf{0}. \quad \forall k \neq t \end{aligned} \tag{7.8}$$

The said projection matrix can be easily constructed by first generating a set of m random orthonormal basis[1] in $\mathbb{R}^m$, then picking $r = \lfloor m/T \rfloor$ of those basis (matrix columns) to form a matrix O_t, and, finally, obtaining the projections as $P_t = O_t O_t^\top$. For different tasks these basis form a disjoint set $\mathcal{P} = \{P_1, \cdots, P_T\}$. If the total number of tasks exceeds the predefined T, then one can potentially dynamically resize the $m \times m$ orthogonal matrix while maintaining the required properties. For example, to make space for $2T$ tasks one can resize the original matrix to $2m \times 2m$ with zero padding, and backup the previous matrix. This would entail dynamically resizing the second last layer of the network. The set $\mathcal{P}$ can be computed offline and stored in a hash table that can be readily fetched given a task identifier. The projection only adds a single matrix multiplication in the forward pass of the network making it as efficient as standard training.

Next, lets examine the effect of the projection step on the backward pass of the backpropagation algorithm. For a task t, the gradient of the objective $\ell(\cdot, \cdot)$ on any intermediate layer h_l can be decomposed using the chain rule as,

$$\begin{aligned} g_l^t = \frac{\partial \ell}{\partial h_l} &= \left(\frac{\partial \ell}{\partial h_L}\right) \frac{\partial h_L}{\partial h_l} = \left(\frac{\partial \phi_t}{\partial h_L} \frac{\partial \ell}{\partial \phi_t}\right) \frac{\partial h_L}{\partial h_l}, \\ &= \left(P_t \frac{\partial \ell}{\partial \phi_t}\right) \prod_{k=l}^{L-1} \frac{\partial h_{k+1}}{\partial h_k} = g_L^t \prod_{k=l}^{L-1} D_{k+1} W_{k+1}, \end{aligned} \tag{7.9}$$

where D_{k+1} is a diagonal matrix representing the Jacobian of the pointwise non-linearity $\sigma_{k+1}(\cdot)$. For a ReLU nonlinearity and assuming that the non-linearity

[1] We generate a random matrix and apply the Gram–Schmidt process offline, before the continual learning experience begins.

Algorithm 10 *Training of ORTHOG-SUBSPACE on sequential data* $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$*, with* $\Theta = \{W_l\}_{l=1}^{L}$ *initialized as orthonormalized matrices,* $\mathcal{P} = \{P_1, \cdots, P_T\}$ *orthogonal projections, learning rate '*α*',* $s = 2$*,* $q = 0.5$*,* $\epsilon = 10^{-8}$*.*

procedure ORTHOG-SUBSPACE($\mathcal{D}, \mathcal{P}, \alpha, s, q, \epsilon$)
  $\mathcal{M} \leftarrow \{\} * T$
  for $t \in \{1, \cdots, T\}$ **do**
    for $(x_t, y_t) \sim \mathcal{D}_t$ **do**
      $k \sim \{1, \cdots, t-1\}$ ▷ Sample a past task from the replay buffer
      $(x_k, y_k) \sim \mathcal{M}_k$ ▷ Sample data from the episodic memory
      $g^t \leftarrow \nabla_{\Theta,\theta}\left(\ell(f(x_t, y_t), P_t)\right)$ ▷ Compute gradient on the current task
      $g^k \leftarrow \nabla_{\Theta,\theta}\left(\ell(f(x_k, y_k), P_k)\right)$ ▷ Compute gradient on the past task
      $g \leftarrow g^t + g^k$
      for $l = \{1, \cdots, L\}$ **do** ▷ Layer-wise update on Stiefel manifold
        $A \leftarrow g_l W_l^\top - W_l g_l^\top$
        $U \leftarrow AW_l$ ▷ Project the gradient onto the tangent space
        $\tau \leftarrow \min(\alpha, 2q/(||W_l|| + \epsilon))$ ▷ Select adaptive learning rate Li et al. [84]
        $Y^0 \leftarrow W_l - \tau U$ ▷ Iterative estimation of the Cayley Transform
        for $i = \{1, \cdots, s\}$ **do**
          $Y^i \leftarrow W_l - \frac{\tau}{2} A(W_l + Y^{i-1})$
        end for
        $W_l \leftarrow Y^s$
      end for
      $\theta \leftarrow \theta - \alpha \cdot g_{L+1}$ ▷ Update the classifier head
      $\mathcal{M}_t \leftarrow (x_t, y_t)$ ▷ Add the sample to a ring buffer
    end for
  end for
  return Θ, θ
end procedure

remains in the linear region during the training [128, 12], we assume the Jacobian matrix to be an identity. It can be seen that for the projected layer L (the second last layer), the gradients of different tasks are orthogonal by construction *i.e.* $g_L^t \perp g_L^{k \neq t}$ (cf. equation 7.8). Hence the gradient interference will be zero at the layer L. However, according to equation 7.9, as the gradients are backpropogated to the previous layers they start to become less and less orthogonal (cf. figure 7.3). This results in interference among different tasks in earlier layers, especially when the network is relatively deep.

Let us rewrite the gradients at the intermediate layer l during the training of task t as a linear transformation of the gradient at the layer L *i.e.* $g_l^t = T(g_L^t)$. According to equation 7.9, and assuming the Jacobian matrix of the non-linearity to be identity

($D_k = I$), this transformation is given by

$$T(u) = u \prod_{k=l}^{L-1} W_{k+1}. \tag{7.10}$$

As noted earlier, $g_L^t \perp g_L^{k \neq t}$ by construction, then to reduce the interference between any g_l^t and $g_l^{k \neq t}$, the transformation $T(\cdot)$ in equation 7.10 needs to be such that it preserves the inner-product between $T(g_L^t)$ and $T(g_L^{k \neq t})$. In other words, $T(\cdot)$ needs to be an isometry (cf. definition 7.2.2). As discussed in section 7.2.2, this is equivalent to ensuring that weight matrices $\{W_l\}_{l=1}^L$ are orthonormal matrices.

We learn orthonormal weights of a neural network by posing the network training as an optimization problem over a Stiefel manifold [1, 23]. More specifically, the network is initialized from random orthonormal matrices. A tiny replay buffer, storing the examples of past tasks ($k < t$), is maintained to compute the gradients $\{g_l^k\}_{l=1}^L$. The gradients on the current task t, $\{g_l^t\}_{l=1}^L$, are computed and weights in each layer l are updated as follows: a) first the effective gradient $g_l = g_l^t + g_l^k$ is projected onto the tangent space at the current estimate of the weight matrix W_l, b) the iterative Cayley Transform equation 7.7 is used to retract the update to the Stiefel manifold. The projection onto the tangent space is carried out using the closed-form described in equation 7.5. The resulting algorithm keeps the network weights orthonormal throughout the continual learning experience while reducing the loss using the projected gradients. Figure 7.3 shows this orthonormality reduces the inner product between the gradients of different tasks. We denote our approach as ORTHOG-SUBSPACE and provide the pseudo-code in algorithm 10.

7.4 Experiments

We now report experiments on continual learning benchmarks in classification tasks.

7.4.1 Continual Learning Benchmarks

We evaluate *average accuracy* equation 7.2 and *forgetting* equation 7.3 on four supervised classification benchmarks. **Permuted MNIST** is a variant of the MNIST dataset of handwritten digits [78] where each task applies a fixed random pixel permutation to the original dataset. **Rotated MNIST** is another variant of MNIST, where each task applies a fixed random image rotation (between 0 and 180 degrees) to the original dataset. Both of the MNIST benchmark contain 23 tasks, each with 10000 samples from 10 different classes. **Split CIFAR** is a variant of the CIFAR-100 dataset [68, 156], where each task contains the data pertaining 5 random classes (without replacement) out of the total 100 classes. **Split miniImageNet** is a variant of the ImageNet dataset [117, 145], containing a subset of images and classes from the original dataset. Similar to Split CIFAR, in Split miniImageNet each task contains the data from 5 random classes (without replacement) out of the total 100 classes. Both CIFAR-100 and miniImageNet contain 20 tasks, each with 250 samples from each of the 5 classes.

Similar to Chaudhry et al. [30], for each benchmark, the first 3 tasks are used for hyper-parameter tuning (grids are available in Appendix section 7.D). The learner can perform multiple passes over the datasets of these three initial tasks. We assume that the continual learning experience begins after these initial tasks and ignore them in the final evaluation.

7.4.2 Baselines

We compare ORTHOG-SUBSPACE against several baselines which we describe next. **Finetune** is a vanilla model trained on a data stream, without any regularization or episodic memory. **ICARL** [110] is a *memory-based* method that uses knowledge-distillation [57] and episodic memory to reduce forgetting. **EWC** [67] is a *regularization-based* method that uses the Fisher Information matrix to record the parameter importance. **VCL** [98] is another *regularization-based* method that uses variational inference to approximate the posterior distribution of the parameters which is regularized during the continual learning experience. **AGEM** [30] is a *memory-based* method similar to [87] that uses episodic memory as an optimization constraint to reduce forgetting on previous tasks. **MER** [111] is another *memory-based* method that uses first-order meta-learning formulation [99] to reduce forgetting on previous tasks. **ER-Ring** [31] is the strongest *memory-based* method that jointly trains new task data with that of the previous tasks. Finally, **Multitask** is an oracle baseline that has access to all data to optimize equation 7.1. It is useful to estimate an upper bound on the obtainable Accuracy equation 7.2.

7.4.3 Code, Architecture and Training Details

Except for VCL, all baselines use the same unified code base which is made publicly available. For VCL [98], the official implementation is used which only works on fully-connected networks. All baselines use the same neural network architectures: a fully-connected network with two hidden layers of 256 ReLU neurons in the MNIST experiments, and a standard ResNet18 [53] in CIFAR and ImageNet experiments. All baselines do a single-pass over the dataset of a task, except for episodic memory that can be replayed multiple times. The task identifiers are used to select the correct output head in the CIFAR and ImageNet experiments. Batch size is set to 10 across experiments and models. A tiny ring memory of 1 example per

Table 7.1: Accuracy equation 7.2 and Forgetting equation 7.3 results of continual learning experiments. When used, episodic memories contain up to one example per class per task. Last row is a multi-task oracle baseline.

Method		Permuted MNIST		Rotated MNIST	
	Memory	Accuracy	Forgetting	Accuracy	Forgetting
Finetune	✗	50.6 (±2.57)	0.44 (±0.02)	43.1 (±1.20)	0.55 (±0.01)
EWC [67]	✗	68.4 (±0.76)	0.25 (±0.01)	43.6 (±0.81)	0.53 (±0.01)
VCL [98]	✗	51.8 (±1.54)	0.44 (±0.01)	48.2 (±0.99)	0.50 (±0.01)
VCL-Random [98]	✔	52.3 (±0.66)	0.43 (±0.01)	54.4 (±1.44)	0.44 (±0.01)
AGEM [30]	✔	78.3 (±0.42)	0.15 (±0.01)	60.5 (±1.77)	0.36 (±0.01)
MER [111]	✔	78.6 (±0.84)	0.15 (±0.01)	68.7 (±0.38)	0.28 (±0.01)
ER-Ring [31]	✔	79.5 (±0.31)	0.12 (±0.01)	70.9 (±0.38)	0.24 (±0.01)
ORTHOG-SUBSPACE (ours)	✗	**86.6** (±0.91)	**0.04** (±0.01)	**80.1** (±0.95)	**0.14** (±0.01)
Multitask		91.3	0.0	94.3	0.0

Method		Split CIFAR		Split miniImageNet	
	Memory	Accuracy	Forgetting	Accuracy	Forgetting
Finetune	✗	42.6 (±2.72)	0.27 (±0.02)	36.1 (±1.31)	0.24 (±0.03)
EWC [67]	✗	43.2 (±2.77)	0.26 (±0.02)	34.8 (±2.34)	0.24 (±0.04)
ICARL [110]	✔	46.4 (±1.21)	0.16 (±0.01)	-	-
AGEM [30]	✔	51.3 (±3.49)	0.18 (±0.03)	42.3 (±1.42)	0.17 (±0.01)
MER [111]	✔	49.7 (±2.97)	0.19 (±0.03)	45.5 (±1.49)	0.15 (±0.01)
ER-Ring [31]	✔	59.6 (±1.19)	0.14 (±0.01)	49.8 (±2.92)	0.12 (±0.01)
ORTHOG-SUBSPACE (ours)	✔	**64.3** (±0.59)	**0.07** (±0.01)	**51.4** (±1.44)	**0.10** (±0.01)
Multitask		71.0	0.0	65.1	0.0

class per task is stored for the memory-based methods. For ORTHOG-SUBSPACE, episodic memory is not used for MNIST experiments, and the same amount of memory as other baselines is used for CIFAR100 and miniImageNet experiments. All experiments run for five different random seeds, each corresponding to a different dataset ordering among tasks, that are fixed across baselines. Averages and standard deviations are reported across these runs.

7.4.4 Results

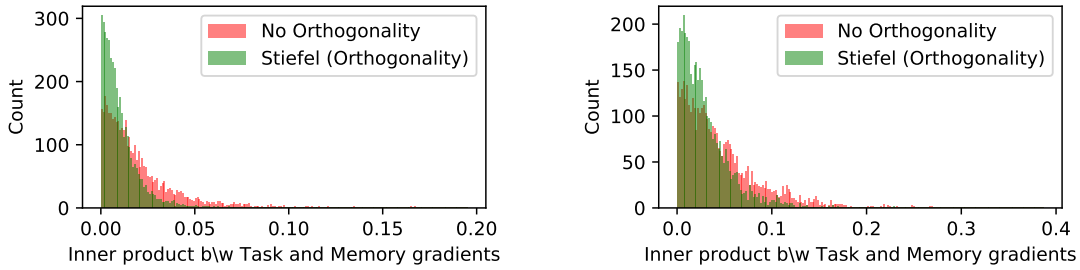
Table 7.1 shows the overall results on all benchmarks. First, we observe that on relatively shallower networks (MNIST benchmarks), even without memory and preservation of orthogonality during the network training, ORTHOG-SUBSPACE

Table 7.2: Systematic evaluation of Projection, Memory and Orthogonalization in ORTHOG-SUBSPACE.

Method			Split CIFAR		Split miniImageNet	
Projection	ER	Stiefel	Accuracy	Forgetting	Accuracy	Forgetting
✔	✗	✗	50.3 ($\pm$2.21)	0.21 ($\pm$0.02)	40.1 ($\pm$2.16)	0.20 ($\pm$0.02)
✗	✔	✗	59.6 ($\pm$1.19)	0.14 ($\pm$0.01)	49.8 ($\pm$2.92)	0.12 ($\pm$0.01)
✔	✔	✗	61.2 ($\pm$1.84)	0.10 ($\pm$0.01)	49.5 ($\pm$2.21)	0.11 ($\pm$0.01)
✔	✔	✔	64.3 ($\pm$0.59)	0.07 ($\pm$0.01)	51.4 ($\pm$1.44)	0.10 ($\pm$0.01)

outperform the strong memory-based baselines by a large margin: $+7.1\%$ and $+9.2\%$ absolute gain in average accuracy, 66% and 42% reduction in forgetting compared to the strongest baseline (ER-Ring), on Permuted and Rotated MNIST, respectively. This shows that learning in orthogonal subspaces is an effective strategy in reducing interference among different tasks. Second, for deeper networks, when memory is used and orthogonality is preserved, ORTHOG-SUBSPACE improves upon ER-Ring considerably: 4.7% and 1.6% absolute gain in average accuracy, 50% and 16.6% reduction in forgetting, on CIFAR100 and miniImageNet, respectively. While we focus on tiny episodic memory, in table 7.3 of Appendix section 7.C, we provide results for larger episodic memory sizes. Our conclusions on tiny memory hold, however, the gap between the performance of ER-Ring and ORTHOG-SUBSPACE gets reduced as the episodic memory size is increased. The network can sufficiently mitigate forgetting by relearning on a large episodic memory.

Table 7.2 shows a systematic evaluation of projection equation 7.8, episodic memory and orthogonalization equation 7.7 in ORTHOG-SUBSPACE. First, without memory and orthogonalization, while a simple projection yields competitive results compared to various baselines (cf. table 7.1), the performance is still a far cry from ER-Ring. However, when the memory is used along with subspace projection one can already see a better performance compared to ER-Ring. Lastly, when the



Layer 15 Layer 10

Figure 7.3: Histogram of inner product of current task and memory gradients in different layers in Split CIFAR. The more left the distribution is the more orthogonal the gradients are and the less the interference is between the current and previous tasks.

orthogonality is ensured by learning on a Stiefel manifold, the model achieves the best performance both in terms of accuracy and forgetting.

Finally, figure 7.3 shows the distribution of the inner product of gradients between the current and previous tasks, stored in the episodic memory. Everything is kept the same except in one case the weight matrices are learned on the Stiefel manifold while in the other no such constraint is placed on the weights. We observe that when the weights remain on the Stiefel manifold, the distribution is more peaky around zero. This empirically validates our hypothesis that by keeping the transformation equation 7.10 isometric, the gradients of different tasks remain near orthogonal to one another in all the layers.

7.5 Related work

In continual learning [112], also called lifelong learning [140], a learner faces a *sequence* of tasks without storing the complete datasets of these tasks. This is in contrast to *multitask learning* [27], where the learner can simultaneously access data from all tasks. The objective in continual learning is to avoid catastrophic forgetting The main challenge in continual learning is to avoid catastrophic forgetting [93, 92, 48] on already seen tasks so that the learner is able to learn new tasks

quickly. Existing literature in continual learning can be broadly categorized into three categories.

First, *regularization approaches* reduce the drift in parameters important for past tasks [67, 3, 98, 156]. For the large number of tasks, the parameter importance measures suffer from brittleness as the locality assumption embedded in the regularization-based approaches is violated [141]. Furthermore, Chaudhry et al. [30] showed that these approaches can only be effective when the learner can perform multiple passes over the datasets of each task – a scenario not assumed in this work. Second, *modular approaches* use different network modules that can be extended for each new task [40, 2, 115, 28, 153, 41]. By construction, modular approaches have zero forgetting, but their memory requirements increase with the number of tasks [118, 80]. Third, *memory approaches* maintain and replay a small episodic memory of data from past tasks. In some of these methods [85, 110], examples in the episodic memory are replayed and predictions are kept invariant by means of distillation [57]. In other approaches [87, 30, 6] the episodic memory is used as an optimization constraint that discourages increases in loss at past tasks. Some works [52, 111, 114, 31, 32] have shown that directly optimizing the loss on the episodic memory, also known as experience replay, is cheaper than constraint-based approaches and improves prediction performance. Recently, Prabhu et al. [106] showed that training at test time, using a greedily balanced collection of episodic memory, improved performance on a variety of benchmarks. Similarly, Javed and White [62] and Beaulieu et al. [18] showed that learning transferable representations via meta-learning reduces forgetting when the model is trained on sequential tasks.

Similar in spirit to our work is OGD Farajtabar et al. [39] where the gradients of each task are learned in the orthogonal space of all the previous tasks' gradients. However, OGD differs significantly from our work in terms of memory and com-

pute requirements. Unlike OGD, where the memory of previous task gradients is maintained, which is equivalent to storing $n_t \times S$ dimensional matrix for each task, where n_t are the number of examples in each task and S is the network size, we only store $m \times r$ dimensional matrix O_t, where m is the dimension of the feature vector ($m \ll S$) and r is the rank of the subspace, and a tiny replay buffer for each task. For large network sizes, OGD is impractical to use. Furthermore, at each training step OGD subtracts the gradient projections from the space spanned by the gradients in memory, whereas we only project the feature extraction layer to a subspace and maintain orthogonality via learning on Stiefel manifolds.

Finally, learning orthonormal weight matrices has been extensively studied in literature. Orthogonal matrices are used in RNNs for avoiding exploding/ vanishing gradient problem [9, 151, 64]. While the weight matrices are assumed to be square in the earlier works, works including [59] considered learning non-square orthogonal matrices (called orthonormal matrices in this work) by optimizing over the Stiefel manifolds. More recently, Li et al. [84] proposed an iterative version of Cayley transform [100], a key component in optimizing over Stiefel manifolds. Whereas optimizing over the Stiefel manifold ensure strict orthogonality in the weights, [63] proposed an algorithm, Singular Value Bounding (SVB), for soft orthogonality constraints. We use strict orthogonality in this work and leave the exploration of soft orthogonality for future research.

7.6 Conclusion

In this chapter, we presented ORTHOG-SUBSPACE, a continual learning method that learns different tasks in orthogonal subspaces. The gradients in the projected layer are kept orthogonal in earlier layers by learning isometric transformations. The isometric transformations are learned by posing the network training as an

optimization problem over the Stiefel manifold. The proposed approach improved considerably over strong memory replay-based baselines in standard continual learning benchmarks of image classification.

Appendix

Section 7.A provides a proof that isometry preserves angles. Section 7.B derives the closed-form of the gradient projection on the tangent space at a point in the Stiefel manifold. Section 7.C gives further experimental results. Section 7.D lists hyperparameters for all experiments.

7.A Isometry Preserves Angles

Theorem 7.A.1. T is an isometry iff it preserves inner products.

Proof. Suppose T is an isometry. Then for any $v, w \in V$,

$$\|T(v) - T(w)\|^2 = \|v - w\|^2$$
$$\langle T(v) - T(w), T(v) - T(w)\rangle = \langle v - w, v - w\rangle$$
$$\|T(v)\|^2 + \|T(w)\|^2 - 2\langle T(v), T(w)\rangle = \|v\|^2 + \|w\|^2 - 2\langle v, w\rangle.$$

Since $\|T(u)\| = \|u\|$ for any u in V, all the length squared terms in the last expression above cancel out and we get

$$\langle T(v), T(w)\rangle = \langle v, w\rangle.$$

Conversely, if T preserves inner products, then

$$\langle T(v - w), T(v - w)\rangle = \langle v - w, v - w\rangle,$$

which implies

$$\|T(v-w)\| = \|v-w\|,$$

and since T is linear,

$$\|T(v)-T(w)\| = \|v-w\|.$$

This shows that T preserves distance. □

7.B Closed-form of Projection in Tangent Space

This section closely follows the arguments of Tagare [137].

Let $\{X \in \mathbb{R}^{n\times p} | X^\top X = I\}$ defines a manifold in Euclidean space $\mathbb{R}^{n\times p}$, where $n > p$. This manifold is called the Stiefel manifold. Let $\mathcal{T}_X$ denotes a tangent space at X.

Lemma 7.B.1. Any $Z \in \mathcal{T}_X$ satisfies:

$$Z^\top X + X^\top Z = 0$$

i.e. $Z^\top X$ is a skew-symmetric $p \times p$ matrix.

Note, that X consists of p orthonormal vectors in $\mathbb{R}^n$. Let $X_\perp$ be a matrix consisting of the additional $n-p$ orthonormal vectors in $\mathbb{R}^n$ *i.e.* $X_\perp$ lies in the orthogonal compliment of X, $X^\top X_\perp = 0$. The concatenation of X and $X_\perp$, $[X\, X_\perp]$ is $n \times n$ orthonormal matrix. Then, any matrix $U \in \mathbb{R}^{n\times p}$ can be represented as: $U = XA + X_\perp B$, where A is a $p \times p$ matrix, and B is a $(n-p) \times p$ matrix.

Lemma 7.B.2. A matrix $Z = XA + X_\perp B$ belongs to the tangent space at a point on Stiefel manifold $\mathcal{T}_X$ iff A is skew-symmetric.

Let $G \in \mathbb{R}^{n \times p}$ be the gradient computed at X. Let the projection of the gradient on the tangent space is denoted by $\pi_{\mathcal{T}_X}(G)$.

Lemma 7.B.3. Under the canonical inner product, the projection of the gradient on the tangent space is given by $\pi_{\mathcal{T}_X}(G) = AX$, where $A = GX^\top - XG^\top$.

Proof. Express $G = XG_A + X_\perp G_B$. Let Z be any vector in the tangent space, expressed as $Z = XZ_A + X_\perp Z_B$, where Z_A is a skew-symmetric matrix according to Lemma 7.B.2. Therefore,

$$\begin{aligned}\pi_{\mathcal{T}_X}(G) &= \text{tr}(G^\top Z), \\ &= \text{tr}((XG_A + X_\perp G_B)^\top (XZ_A + X_\perp Z_B)), \\ &= \text{tr}(G_A^\top Z_A + G_B^\top Z_B). \end{aligned} \tag{7.11}$$

Writing G_A as $G_A = \text{sym}(G_A) + \text{skew}(G_A)$, and plugging in equation 7.11 gives,

$$\pi_{\mathcal{T}_X}(G) = \text{tr}(\text{skew}(G_A)^\top Z_A + G_B^\top Z_B). \tag{7.12}$$

Let $U = XA + X_\perp B$ is the vector that represents the projection of G on the tangent space at X. Then,

$$\begin{aligned}\langle U, Z \rangle_c &= \text{tr}(U^\top (I - \frac{1}{2} XX^\top) Z), \\ &= \text{tr}((XA + X_\perp B)^\top (I - \frac{1}{2} XX^\top)(XZ_A + X_\perp Z_B)), \\ &= \text{tr}(\frac{1}{2} A^\top Z_A + B^\top Z_B) \end{aligned} \tag{7.13}$$

By comparing equation 7.12 and equation 7.13, we get $A = 2\text{skew}(G_A)$ and $B = G_B$.

Thus,

$$
\begin{aligned}
U &= 2X\text{skew}(G_A) + X_\perp G_B, \\
&= X(G_A - G_A^\top) + X_\perp G_B, \quad \because \text{skew}(G_A) = \frac{1}{2}(G_A - G_A^\top) \\
&= XG_A - XG_A^\top + G - XG_A, \quad \because G = XG_A + X_\perp G_B \\
&= G - XG_A^\top, \\
&= G - XG^\top X, \quad \because G_A = X^\top G, \\
&= GX^\top X - XG^\top X, \\
&= (GX^\top - XG^\top)X
\end{aligned}
$$

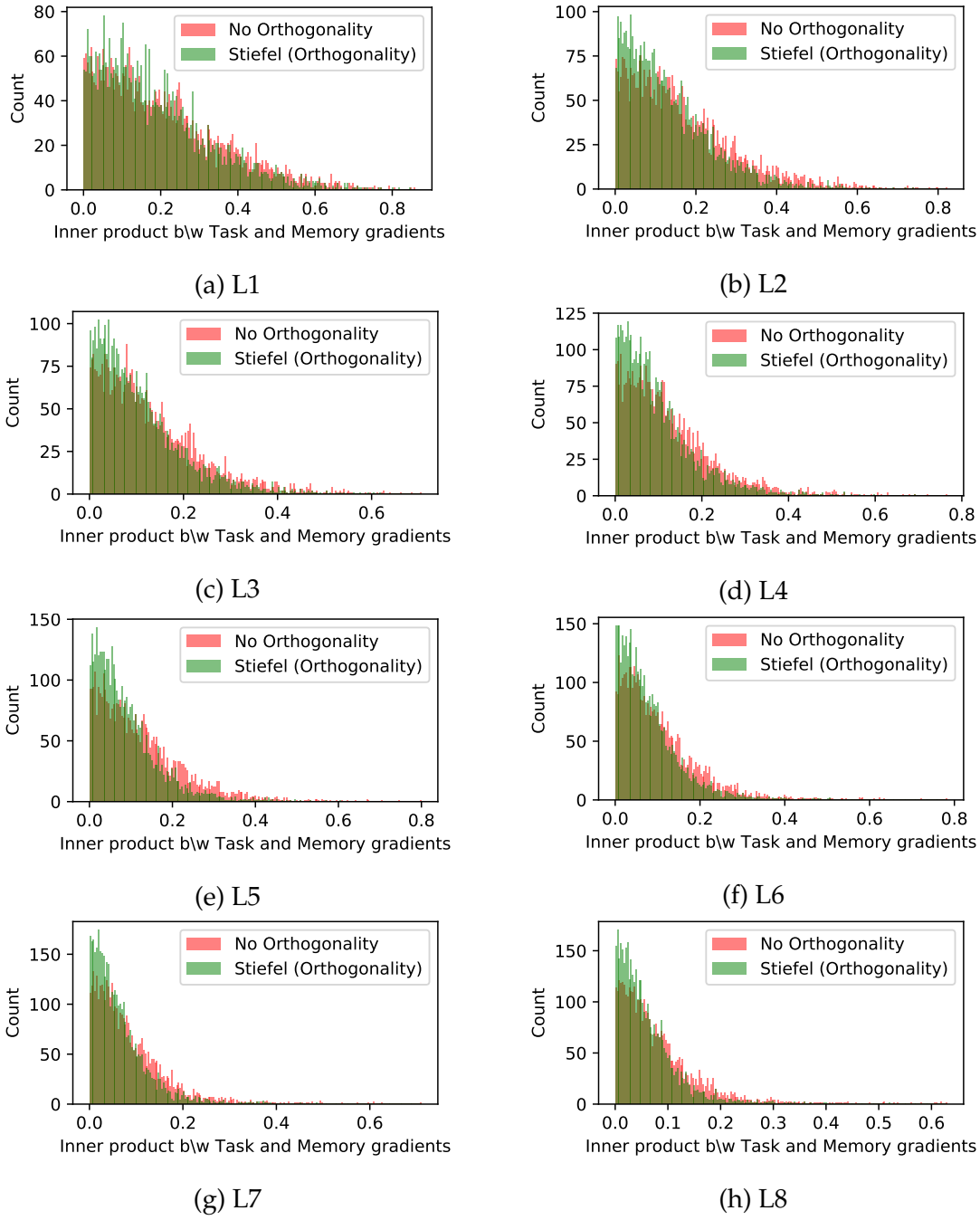
□

7.C More Results

Table 7.3: Accuracy equation 7.2 and Forgetting equation 7.3 results of continual learning experiments for larger episodic memory sizes. 2, 3 and 5 samples per class per task are stored, respectively. Top table is for Split CIFAR. Bottom table is for Split miniImageNet.

Method	Accuracy			Forgetting		
	2	3	5	2	3	5
AGEM	52.2 (±2.59)	56.1 (±1.52)	60.9 (±2.50)	0.16 (±0.01)	0.13 (±0.01)	0.11 (±0.01)
ER-Ring	61.9 (±1.92)	64.8 (±0.77)	67.2 (±1.72)	0.11 (±0.02)	0.08 (±0.01)	0.06 (±0.01)
ORTHOG-SUBSPACE	64.7 (±0.53)	66.8 (±0.83)	67.3 (±0.98)	0.07 (±0.01)	0.05 (±0.01)	0.05 (±0.01)

Method	Accuracy			Forgetting		
	2	3	5	2	3	5
AGEM	45.2 (±2.35)	47.5 (±2.59)	49.2 (±3.35)	0.14 (±0.01)	0.13 (±0.01)	0.10 (±0.01)
ER-Ring	51.2 (±1.99)	53.9 (±2.04)	56.8 (±2.31)	0.10 (±0.01)	0.09 (±0.02)	0.06 (±0.01)
ORTHOG-SUBSPACE	53.4 (±1.23)	55.6 (±0.55)	58.2 (±1.08)	0.07 (±0.01)	0.06 (±0.01)	0.05 (±0.01)



(a) L1 (b) L2

(c) L3 (d) L4

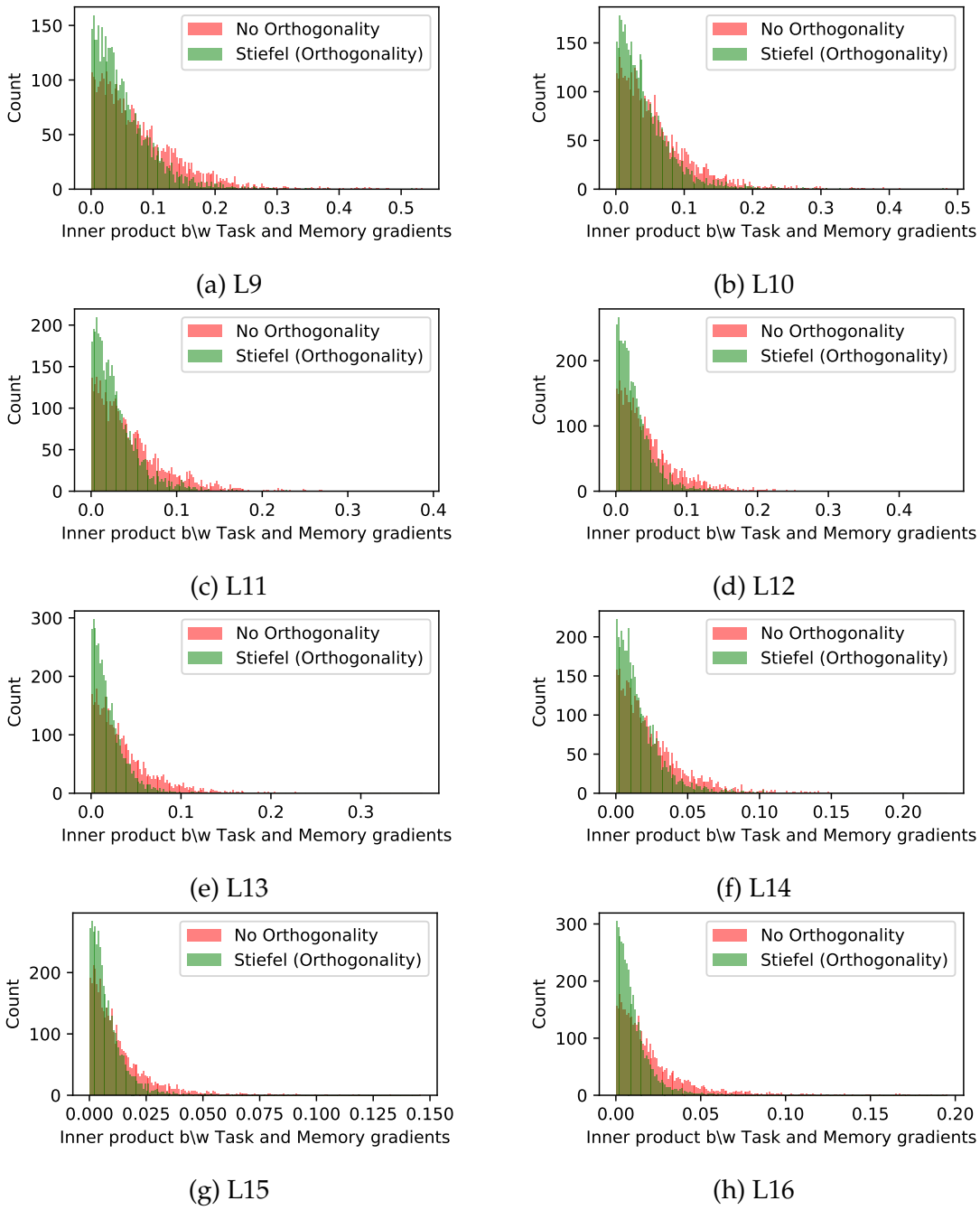
(e) L5 (f) L6

(g) L7 (h) L8

Figure 7.4: Histogram of inner product of current task and memory gradients in all layers in Split CIFAR.

7.D Hyperparameters

Table 7.4 lists the best hyperparameters for each experiment.



(a) L9 (b) L10

(c) L11 (d) L12

(e) L13 (f) L14

(g) L15 (h) L16

Figure 7.5: Histogram of inner product of current task and memory gradients in all layers in Split CIFAR.

Table 7.4: Hyperparameters for all the baselines.

Method	Dataset	Hyperparameters
Multitask	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
Finetune	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
EWC	Permuted MNIST	learning rate (0.1), regularization (10)
	Rotated MNIST	learning rate (0.1), regularization (10)
	Split CIFAR-100	learning rate (0.03), regularization (10)
	Split miniImageNet	learning rate (0.03), regularization (10)
A-GEM	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
MER	Permuted MNIST	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (10)
	Rotated MNIST	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (10)
	Split CIFAR-100	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (5)
	Split miniImageNet	learning rate (0.03), within batch meta lr (0.1), current batch lr multiplier (5)
ER-Ring	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.03)
	Split miniImageNet	learning rate (0.03)
ORTHOG-SUBSPACE	Permuted MNIST	learning rate (0.1)
	Rotated MNIST	learning rate (0.1)
	Split CIFAR-100	learning rate (0.4)
	Split miniImageNet	learning rate (0.2)

Chapter 8

Conclusions

This thesis has covered continual learning algorithms based on network parameter regularization, episodic memory, nested optimization, and subspace separation. Alongside, the thesis has also discussed different training and evaluation regimes in continual learning. Efficiency of learning – measured in terms of faster adaptation, compute and small episodic memories – remained the focus of all chapters of this thesis. We now summarize the contributions made in the main chapters 3–7 in section 8.1, look at some future challenges in section 8.2, and finally end the thesis with some concluding remarks in section 8.3.

8.1 Summary of contributions

Chapter 3: Parameter Regularization-based Continual Learning: Riemannian Walk.

In this chapter, we derived a parameter regularization-based continual learning objective by reducing the KL-divergence between the conditional likelihoods of two successive tasks. Regularizing the model likelihood between previous and new tasks allows the continual learner to preserve the inherent properties of the model

on previous tasks. By utilizing the well-known approximation of the KL-divergence using Taylor series [7, 102], we showed that our regularization objective gets the similar form as that of the Kirkpatrick et al. [67]. However, unlike Kirkpatrick et al. [67], where a curvature information of each task is stored, we maintain a single curvature (Fisher information matrix) computed as a running average of previous curvatures. We further add optimization path-specific scalars to the diagonals of the Fisher information matrix that allows our regularization objective to respond to the training trajectory. Note in Kirkpatrick et al. [67], the regularization objective only depends on the minimum obtained after training each task. All in all, our method can be thought of as a generalization of Kirkpatrick et al. [67] and Zenke et al. [156].

In addition to a continual learning algorithm (RWALK), the chapter also discussed different evaluation protocols for continual learning and showed that evaluation setups where no task information is provided at test time (a single-head setup) are much harder than multihead setups (where a task information is provided at test time). The chapter also discussed the importance of samples from previous tasks in handling intransigence in regularization-based continual learners. We note that the methods based solely on episodic memory were not discussed in this chapter and they were the subjects of the next chapters.

The proposed algorithm (RWALK) achieved a better performance in both the single- and multi-head evaluation setups. We also analyzed the stability-plasticity dilemma encoded in the interplay of forgetting and intransigence, and showed that RWALK achieved the best trade-off among all continual learning algorithms at the time of publication of the work.

Chapter 4: Efficient Continual Learning: Averaged Gradient Episodic Memory.

In the previous chapter, we focused on a regularization-based continual learning algorithm and considered a batch mode training setup whereby a model can store the complete dataset of a task offline and perform multiepoch training. In this (and subsequent) chapter(s), we shift our focus to episodic memory-based continual learning algorithms that are arguably more efficient and consider an online setup where the model can only do a single pass through the dataset of each task.

In this chapter, first, we proposed a more realistic continual learning protocol where not only the model can just do a single pass through the data, but also the model was not allowed to revisit any task for the tuning of hyperparameters. Along with this learning protocol, we proposed a metric (LCA) that computes the learning speed of a continual learner allowing us to quantify the transfer from previous tasks to new ones.

We then proposed an efficient variant of the model suggested by Lopez-Paz and Ranzato [87] whereby the episodic memory of previous tasks was used as a set of optimization constraints preventing the loss of those tasks from increasing as a new is learned. Our algorithmic contribution was to simplify the optimization problem of Lopez-Paz and Ranzato [87], a quadratic program (QP), by using a single constraint computed as the average of gradients on the episodic memory. By not solving the QP at each training step, our proposed algorithm (A-GEM) is not only memory efficient but also two orders of magnitude faster than that of Lopez-Paz and Ranzato [87]. We also showed that our algorithm incurred less constraint violations than that of Lopez-Paz and Ranzato [87] further improving the training speed and the quality of the optimum for each task.

Finally, to improve the forward transfer in continual learning algorithms, we proposed to use more elaborate task descriptions as training signals. Specifically,

we use the concatenated class attributes as the task descriptors and used a joint embedding model that learns a shared embedding space between image features and attribute embeddings. We experimentally showed that a) A-GEM achieved the best trade-off in terms of performance and efficiency, and b) the joint embedding of input and task attributes improves the forward transfer of continual learning algorithms across the board. We also experimentally showed that regularization-based continual learning algorithms, such as EWC [67], require highly over-parameterized architectures and batch mode training (multi-epoch training) to perform well.

Chapter 5: Continual Learning by Directly Replaying the Episodic Memory.

This chapter simplified the optimization objective developed in the previous chapter. Whereas in the previous chapter the episodic memory was used as an optimization constraint, in this chapter episodic memory was directly used in the loss function. The resulting, much simplified algorithm, experience replay (ER), generalized better than A-GEM for small episodic memories.

This finding was rather surprising as the common intuition is that over parameterized models tend to overfit to small data when trained repeatedly. To analyze the role of overfitting (or memorizing small data) in continual learning, we conducted controlled experiments where the relatedness of the two tasks was carefully chosen. Our experiments suggested that memorization in continual learning leads to better generalization and that the data of incoming similar tasks act as a strong but implicit data-dependent regularizer, preventing the memorization of small memories from deteriorating the performance. Experimentally, we showed ER to be the simplest and the strongest continual learning algorithm.

Chapter 6: Synthesizing and Anchoring Memory Samples for Continual Learning.

This chapter was built on the previous chapter where ER was the strongest baseline. The question we asked in this chapter was whether we can synthesize the data samples that are stored in the episodic memory? We presented an algorithm, hindsight anchor learning (HAL), that allowed for the generation of synthesized samples, called anchors, for each task. The objective of the algorithm was to generate samples that would undergo maximum forgetting throughout the continual learning experience. These samples were generated using gradient ascent in the image space by maximizing the forgetting loss in the replay buffer – in this sense, we said that the forgetting loss was maximized in hindsight. We showed that these anchors lie close to the decision boundaries acting as support vectors for the corresponding hyperplanes.

The chapter also presented a nested optimization objective for continual learning, called anchoring objective, that regularizes the change of the model on the synthesized anchoring points. It was also shown in the chapter that the anchoring objective maximizes the inner product between the gradients of different tasks. This maximization of the inner product has been shown to improve the forward transfer in a meta-learning setting [99, 42]. Experimentally, we showed that HAL improves upon the experience replay both in terms of accuracy and forgetting for the same amount of episodic memory.

Chapter 7: Continual Learning in Low-rank Orthogonal Subspaces.

In all previous chapters, the continual learning algorithms were learning in the same feature (vector) space. In this chapter, we proposed to learn different tasks in nonoverlapping vector subspaces. The idea was that if the distributed repre-

sentational overlap in the network layers can be reduced, it should reduce the interference between different tasks.

To achieve this, we proposed to project the features of each task into a subspace by a task-specific projection matrix such that for different tasks these matrices are orthogonal to each other. A trivial solution would then have been to apply similar projections at each layer of the network. Instead, we only applied this projection at the second last layer (a.k.a. feature extraction layer) and modified the backward pass of the backpropagation algorithm such that the interference between the gradients coming from the nonoverlapping subspaces is minimized. For this, we proposed to learn the isometries in the form of orthogonal weights of the neural network. This is achieved by posing the network training as an optimization problem over the Stiefel manifolds for which we used well-known results from the optimization in the Riemannian manifolds literature. The resulting algorithm reduced the interference between the representations of different tasks by keeping their gradients close to orthogonal. Experimentally, we showed that learning tasks in orthogonal subspaces effectively reduces forgetting by a significant margin compared to the other baselines.

8.2 Future challenges

We now highlight some areas where further breakthroughs are required for continual learning settings to become ubiquitous in machine learning.

Realistic benchmarks

We argued in this thesis that for machine learning to be deployed in-the-wild, the models need to learn continually from their environments. While a continual learner should be a general purpose algorithm capable of working in and learning

from any environment, the need for effective benchmarks remains. Most of the progress in machine learning, computer vision, and natural language processing in the last two decades is driven by the availability of such benchmarks that can quantify the efficacy of different algorithms for a given problem.

Continual learning benchmarks used in this thesis, and elsewhere in the present literature, are contrived simplifications of a true continual learning setup. In most cases, these benchmarks are created by modifying the fixed (static) computer vision or natural language processing datasets to create a sequence of a limited number of tasks. Most of the goals of continual learning, such as quick adaptation, faster convergence, positive backward transfer *etc*., are not observed in these limited datasets. More focus needs to be put in creating large continual learning benchmarks. We identify some of the desirable properties of such a benchmark in the following:

- **Large number of tasks.** Current continual learning benchmarks are limited to a small number of tasks. In most cases, this number is less than 50. For a continual learning algorithm to exhibit the desirable properties of faster adaptation or forward transfer, our belief is that it needs to learn from tens of thousands of tasks. These tasks need not all be supervised. After all, humans slowly learns from myriad of unsupervised tasks (observations without supervision) in their first few months [89].

- **Gradually changing tasks.** In current continual learning benchmarks either the input distribution of tasks changes abruptly (*e.g*. Permuted MNIST) or the output distribution undergoes a sudden change (*e.g*. Split CIFAR, ImageNet *etc*.). We believe that this sudden change in tasks is not only unnatural, as the real world undergoes smooth transitions, but also contributes to more forgetting. Contrary to this, Smith and Slone [133] argued that humans learn from skewed, ordered, and slowly varying transitions of the visual environment. We need to generate benchmarks that follow similar gradual

transitions. Perhaps, we may also want to introduce a curriculum in our benchmarks – arranging the tasks in an increasing order of difficulty. Towards this Chevalier-Boisvert et al. [34] proposed BabyAI, a grounded language learning suite with 19 levels of increasing difficulty.

- **Compositional tasks.** We explored the composition of task descriptors in section 4.5 and showed that it led to faster (more sample efficient) learning of new tasks. While the objective of each task was different *i.e.*, for example, to classify different bird categories, the composition among them was artificially introduced by concatenating the class attributes as descriptors. We now argue that such composition should be inherent in the nature of tasks and that a continual learning benchmark, in addition to creating new and completely different tasks should also create tasks that are compositions of the previous ones. Examples of such tasks could be natural language tasks exploiting the compositional syntactic structures in human language *e.g.* robot navigation tasks guided by human language.

Statistical learning theory for non-iid data

The learning tools for continual learning that are adopted in this thesis, and elsewhere in the present literature, are heavily borrowed from statistical learning theory [143]. The most fundamental of these tools is empirical risk minimization (ERM) as introduced in section 2.1. The two main assumptions made in statistical learning are 1) that future is related to the past *i.e.* training and test sets are drawn from the same fixed probability distribution, and 2) that the data generating process samples both the train and test sets iid from this distribution.

Both of these assumptions do not hold for a continual learner – not only can one not predicate the future tasks at present, meaning the distribution of tasks can change, but also the sequential arrival of tasks does not follow the iid assumption.

The generalization bounds [143] one has for ERM do not hold for this setting. Therefore, one needs an alternative learning theory for non-iid data to access how well continual learning algorithm is learning, or perhaps more fundamentally, what it means to *learn* in a continual learning setting. Of course, from the *no free lunch theorem* one has to make some assumptions on the data distribution, otherwise one will have no confidence in the generalization. As of this writing, we do not have a clear idea of how such a theory would look.

Alternatives to backpropagation

Backpropagation [116] is a fundamental tool for training deep neural networks. From the perspective of continual learning, the global update of backpropagation (changing all parameters of the network) for *each* data point is detrimental to the previously learned knowledge and increases forgetting catastrophically. We argue for the variant of backpropagation where the weight updates are more local and credit assignment is done in a task or example-specific way such that the activation of a neuron and its neighbours is taken into account while updating the weights. This would ensure that only sufficiently active neurons are updated. Such alternatives to backpropagation could reduce forgetting in the continual learning setting.

Causal representation for continual learning

Current machine learning models, say vision-based models, learn directly from raw sensory data (*e.g.* pixels in the image). While this learning setting is useful in terms of end-to-end training of a model, it is also prone to making large changes in the model when only small factors of variation are changed. For example, a model that learns directly from image pixels needs to make quite a few weight updates to adjust to images in the dark from images in the light. This not only requires

many weight updates, leading to slow convergence, but also a large number of observations to adapt to the changed setting. This frequent weight updates also make the machine learning models more prone to forgetting previous information.

However, if the learning is done in the right space, the changes in the input distribution would be more localized. Causal representations or variables provide one such space. Discovering causal representations from data is an active area of research [105, 86, 35]. Recently, Bengio et al. [21] argued that good representations lead to faster convergence, therefore, one can recover such representations by encouraging fast adaptation through a meta-learning objective. From a continual learning perspective, such representations are less prone to forgetting as they require less number of weight updates.

8.3 Concluding remarks

Deep learning has seen tremendous success over the last decade owing to the availability of large datasets and computing power. However, the deep learning models do not learn efficiently and are restricted to single task domains. On the other hand, in the real world, we need models that not only learn efficiently but can also be used for multiple tasks.

Continual learning provides an efficient way to learn from a sequence of tasks, leveraging previous knowledge to adapt to new tasks quickly. This thesis proposed different continual learning algorithms that can learn efficiently from data. However, as discussed in the previous section, there are still challenges in continual learning that require new benchmarks, perhaps a different learning theory, and a new way of representing and updating the knowledge in neural networks. Further advances in these areas will make continual learning setting ubiquitous in machine learning.

Bibliography

[1] Pierre-Antoine Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *CVPR*, pages 7120–7129, 2017.

[3] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, 2018.

[4] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9357-online-continual-learning-with-maximal-interfered-retrieval.pdf.

[5] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *CVPR*, pages 11254–11263, 2019.

[6] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Online continual learning with no task boundaries. *arXiv preprint arXiv:1903.08671*, 2019.

[7] Shun'ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 1998.

[8] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[9] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[10] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[11] Anurag Arnab and Philip HS Torr. Pixelwise instance segmentation with a dynamically instantiated network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 441–450, 2017.

[12] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

[13] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural ma-

chine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[15] Jean M Barnes and Benton J Underwood. " fate" of first-list associations in transfer theory. *Journal of experimental psychology*, 58(2):97, 1959.

[16] Marco Baroni, Armand Joulin, Allan Jabri, Germàn Kruszewski, Angeliki Lazaridou, Klemen Simonic, and Tomas Mikolov. Commai: Evaluating the first steps towards a useful general ai. *arXiv preprint arXiv:1701.08954*, 2017.

[17] Amy Bearman, Olga Russakovsky, Vittorio Ferrari, and Fei-Fei Li. What's the point: Semantic segmentation with point supervision. In *ECCV*, 2016.

[18] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. *arXiv preprint arXiv:2002.09571*, 2020.

[19] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Citeseer, 1990.

[20] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[21] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. *arXiv preprint arXiv:1901.10912*, 2019.

[22] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016.

[23] Silvere Bonnabel. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013.

[24] George Boole. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*, volume 2. Walton and Maberly, 1854.

[25] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[26] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[27] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[28] Michael Chang, Abhishek Gupta, Sergey Levine, and Thomas L. Griffiths. Automatically composing representation transformations as a means for generalization. In *ICML workshop Neural Abstract Machines and Program Induction v2*, 2018.

[29] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *ECCV*, 2018.

[30] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019.

[31] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486*, 2019.

[32] Arslan Chaudhry, Albert Gordo, Puneet K Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. *arXiv preprint arXiv:2002.08165*, 2020.

[33] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected. In *ICLR*, 2015.

[34] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *arXiv preprint arXiv:1810.08272*, 2018.

[35] Ishita Dasgupta, Jane Wang, Silvia Chiappa, Jovana Mitrovic, Pedro Ortega, David Raposo, Edward Hughes, Peter Battaglia, Matthew Botvinick, and Zeb Kurth-Nelson. Causal reasoning from meta-reinforcement learning. *arXiv preprint arXiv:1901.08162*, 2019.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[37] Mohamed Elhoseiny, Ahmed Elgammal, and Babak Saleh. Write a classifier: Predicting visual classifiers from unstructured text. *IEEE TPAMI*, 39(12): 2539–2553, 2017.

[38] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. In *IJCV*, 2010.

[39] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. *arXiv preprint arXiv:1910.07104*, 2019.

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

[41] Leslie P. Kaelbling Ferran Alet, Tomas Lozano-Perez. Modular meta-learning. *arXiv preprint arXiv:1806.10166v1*, 2018.

[42] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[43] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1146–1155. JMLR. org, 2017.

[44] Robert M French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Proceedings of the 13th annual cognitive science society conference*, volume 1, pages 173–178, 1991.

[45] Robert M French. Catastrophic interference in connectionist networks. *Encyclopedia of Cognitive Science*, 1:431–435, 2003.

[46] Robert M French and André Ferrara. Modeling time perception in rats: Evidence for catastrophic interference in animal learning. In *Proceedings of the 21st Annual Conference of the Cognitive Science Conference*, pages 173–178. Citeseer, 1999.

[47] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. Deep learning. *MIT press Cambridge*, 2016.

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

[49] Stephen T Grossberg. *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control*, volume 70. Springer Science & Business Media, 2012.

[50] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *ICML*, 2016.

[51] John Haugeland. *Artificial intelligence: The very idea*. MIT press, 1989.

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

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[54] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[55] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu. Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201*, 2019.

[56] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[57] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS*, 2014.

[58] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871*, 2018.

[59] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[60] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. *arXiv preprint arXiv:1802.10269*, 2018.

[61] David Isele, Mohammad Rostami, and Eric Eaton. Using task features for zero-shot knowledge transfer in lifelong learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1620–1626. AAAI Press, 2016. ISBN 978-1-57735-770-4.

[62] Khurram Javed and Martha White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems*, pages 1820–1830, 2019.

[63] Kui Jia, Shuai Li, Yuxin Wen, Tongliang Liu, and Dacheng Tao. Orthogonal deep neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[64] Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1733–1741. JMLR. org, 2017.

[65] Nan Rosemary Ke, Olexa Bilaniuk, Anirudh Goyal, Stefan Bauer, Hugo Larochelle, Bernhard Schölkopf, Michael C Mozer, Chris Pal, and Yoshua

Bengio. Learning neural causal models from unknown interventions. *arXiv preprint arXiv:1910.01075*, 2019.

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

[67] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *PNAS*, 2016.

[68] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *https://www.cs.toronto.edu/ kriz/cifar.html*, 2009.

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

[70] John K Kruschke. Alcove: an exemplar-based connectionist model of category learning. *Psychological review*, 99(1):22, 1992.

[71] Solomon Kullback and Richard Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 1951.

[72] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387, 2016.

[73] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander

Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV*, 2020.

[74] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 951–958. IEEE, 2009.

[75] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.

[76] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017.

[77] Nicolas Le Roux, Manzagol Pierre-Antoine, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *NIPS*, 2007.

[78] Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[79] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[80] Jeongtae Lee, Jaehong Yun, Sungju Hwang, and Eunho Yang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[81] John M Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006.

[82] Sang-Woo Lee, Jin-Hwa Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, 2017.

[83] Stephan Lewandowsky. Gradual unlearning and catastrophic interference: A comparison of distributed architectures. *Relating theory and data: Essays on human memory in honor of Bennet B. Murdock*, pages 445–476, 1991.

[84] Jun Li, Fuxin Li, and Sinisa Todorovic. Efficient riemannian optimization on the stiefel manifold via the cayley transform. In *International Conference on Learning Representations*, 2020.

[85] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *ECCV*, pages 614–629, 2016.

[86] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124, 2019.

[87] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continuum learning. In *NIPS*, 2017.

[88] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[89] John Macnamara. Names for things, 1982.

[90] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015.

[91] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[92] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[93] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

[94] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[95] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[96] Tsendsuren Munkhdalai and Hong Yu. Neural semantic encoders. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 1, page 397. NIH Public Access, 2017.

[97] Jacob MJ Murre. *Learning and categorization in modular neural networks*. Psychology Press, 2014.

[98] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *ICLR*, 2018.

[99] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999, 2018*, 2018.

[100] Yasunori Nishimori and Shotaro Akaho. Learning algorithms utilizing quasi-geodesic flows on the stiefel manifold. *Neurocomputing*, 67:106–135, 2005.

[101] Dim P Papadopoulos, Alasdair DF Clarke, Frank Keller, and Vittorio Ferrari. Training object class detectors from eye tracking data. In *European conference on computer vision*, pages 361–376. Springer, 2014.

[102] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *ICLR*, 2014.

[103] Arijit Patra and J Alison Noble. Multi-anatomy localization in fetal echocardiography videos. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 1761–1764. IEEE, 2019.

[104] Judea Pearl. *Causality*. Cambridge university press, 2009.

[105] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference*. The MIT Press, 2017.

[106] Ameya Prabhu, Phil H. S. Torr, and P. K. Dokania. GDumb: A simple approach that questions our progress in continual learning. In *ECCV*, 2020.

[107] Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[108] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2016.

[109] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017.

[110] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, 2017.

[111] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *ICLR*, 2019.

[112] Mark B Ring. Child: A first step towards continual learning. *Machine Learning*, 28(1):77–104, 1997.

[113] Stephen Roller, Y-Lan Boureau, Jason Weston, Antoine Bordes, Emily Dinan, Angela Fan, David Gunning, Da Ju, Margaret Li, Spencer Poff, et al. Open-domain conversational agents: Current progress, open problems, and future directions. *arXiv preprint arXiv:2006.12442*, 2020.

[114] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning. *CoRR*, abs/1811.11682, 2018. URL http://arxiv.org/abs/1811.11682.

[115] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *ICLR*, 2018.

[116] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[117] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

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

[119] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010.

[120] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[121] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[122] Juergen Schmidhuber. A general method for incremental self-improvement and multi-agent learning. In *Evolutionary Computation: Theory and Applications*, pages 81–123. World Scientific, 1999.

[123] Bernhard Schölkopf. Causality for machine learning. *arXiv preprint arXiv:1911.10500*, 2019.

[124] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[125] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[126] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress and compress: A scalable framework for continual learning. In *International Conference in Machine Learning*, 2018.

[127] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *ICML*, 2018.

[128] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. *arXiv preprint arXiv:1711.02114*, 2017.

[129] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NIPS*, 2017.

[130] Kurt Shuster, Eric Michael Smith, Da Ju, and Jason Weston. Multi-modal open-domain dialogue. *arXiv preprint arXiv:2010.01082*, 2020.

[131] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[132] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR*, 2014.

[133] Linda B Smith and Lauren K Slone. A developmental approach to machine learning? *Frontiers in psychology*, 8:2124, 2017.

[134] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *ALT*, pages 13–31. Springer, 2007.

[135] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackerman, et al. A deep learning approach to antibiotic discovery. *Cell*, 2020.

[136] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. *The 10th International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[137] Hemant D Tagare. Notes on optimization on stiefel manifolds. In *Technical report, Technical report*. Yale University, 2011.

[138] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10781–10790, 2020.

[139] Alexander V Terekhov, Guglielmo Montone, and J Kevin O'Regan. Knowledge transfer in deep block-modular neural networks. In *Conference on Biomimetic and Biohybrid Systems*, pages 268–279, 2015.

[140] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

[141] Michalis K Titsias, Jonathan Schwarz, Alexander G de G Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning using gaussian processes. *arXiv preprint arXiv:1901.11356*, 2019.

[142] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.

[143] Vladimir Vapnik. *Statistical learning theory*. Wiley, 1998.

[144] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[145] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NIPS*, pages 3630–3638, 2016.

[146] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[147] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[148] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1328–1338, 2019.

[149] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A. Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

[150] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[151] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les

Atlas. Full-capacity unitary recurrent neural networks. In *Advances in neural information processing systems*, pages 4880–4888, 2016.

[152] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning-a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[153] Ju Xu and Zhanxing Zhu. Reinforced continual learning. In *arXiv preprint arXiv:1805.12369v1*, 2018.

[154] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[155] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[156] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017.

[157] Ji Zhang, Yannis Kalantidis, Marcus Rohrbach, Manohar Paluri, Ahmed Elgammal, and Mohamed Elhoseiny. Large-scale visual relationship understanding. *arXiv preprint arXiv:1804.10660*, 2018.

[158] Yang-Yang Zheng, Jian-Lei Kong, Xue-Bo Jin, Xiao-Yi Wang, Ting-Li Su, and Min Zuo. Cropdeep: The crop vision dataset for deep-learning-based classification and detection in precision agriculture. *Sensors*, 19(5):1058, 2019.

[159] Xinjie Zhou, Xiaojun Wan, and Jianguo Xiao. Attention-based lstm network

for cross-lingual sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 247–256, 2016.