Preventing Forgetting and Promoting Transfer in Continual Learning

Lucas Pagé-Caccia

School of Computer Science
McGill University
Montreal, Canada

December 2023

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

In this thesis, we explore the development of methods for *efficient* continual learning, to enable the sequential acquisition of knowledge by artificial intelligence (AI) systems as they adapt to new information over time. In the first chapters, we investigate approaches to address challenges posed by rehearsal-based methods in overcoming catastrophic forgetting, a major obstacle faced by deep neural networks when learning sequentially. We propose a Maximally Interfered Retrieval (MIR) method to optimize the efficiency of the replay step, which retrieves samples from the replay buffer whose prediction will be the most impacted by the foreseen parameter update on the newly received data. We then introduce Adaptive Quantization Modules (AQMs) for efficient memory storage in online settings. AQM leverages discrete auto-encoders to control variation in the compression ability of the module at any given stage of learning, while ensuring that representations derived from earlier encoder states are usable by later decoder states. Next, we attempt to gain a deeper understanding on what causes catastrophic forgetting in standard online class-incremental settings. We show that replay methods cause the newly added classes' representations to overlap significantly with the previous classes, leading to highly disruptive parameter updates, and propose solutions to mitigate this. The second part of the thesis examines strategies for enabling forward transfer while minimizing the compute cost of adapting models in sequential settings. We first formalize this setting, and tackle the question of when a model should be retrained upon receiving new data, and what architectural and optimization choices are best suited for this setting. We conclude by investigating the use of modular methods in natural language tasks to enable sample-efficient generalization, while remaining highly parameter efficient during transfer to new tasks. In summary, this thesis contributes to the ongoing efforts to develop versatile and efficient machine learning models for sequential adaptation and highlights the importance of addressing various aspects of efficiency in continual learning.

Résumé

Dans cette thèse, nous explorons le développement de méthodes d'apprentissage continu *efficaces*, permettant l'acquisition séquentielle de connaissances par des systèmes d'intelligence artificielle (IA) lorsqu'ils sont confrontés à de nouvelles informations au fil du temps. Dans les premiers chapitres, nous étudions des approches addressant les défis posés par les méthodes basées sur la répétition d'anciennes données pour surmonter l'oubli catastrophique, un obstacle majeur rencontré par les réseaux de neurones profonds lors de l'apprentissage séquentiel. Nous proposons une méthode de Récupération Maximale des données interférées (MIR) pour optimiser l'efficacité de l'étape de relecture, qui récupère des échantillons de la mémoire externe dont la prédiction sera la plus impactée par la mise à jour des paramètres prévue sur les nouvelles données reçues. Nous introduisons ensuite des Modules de Quantification Adaptative (AQM) pour un stockage efficace de la mémoire dans des contextes d'apprentissage en ligne. AQM utilise des auto-encodeurs discrets pour contrôler la variation de la capacité de compression du module à n'importe quel stade de l'apprentissage, tout en garantissant que les représentations dérivées des états précédents de l'encodeur sont réutilisables par les états ultérieurs du décodeur. Ensuite, nous cherchons à mieux comprendre les causes de l'oubli catastrophique dans les contextes standard d'apprentissage en ligne avec ajout de classes. Nous montrons que les méthodes de répétition de données provoquent un chevauchement important des représentations des nouvelles classes avec celles des classes précédentes, conduisant à des mises à jour de paramètres très perturbatrices, et proposons des solutions pour atténuer ce problème. La deuxième partie de la thèse examine des stratégies pour permettre un transfert de connaissances vers des tâches futures, tout en minimisant le coût de calcul de l'adaptation de grands modèles dans des contextes séquentiels. Nous formalisons d'abord ce contexte et abordons la question de savoir quand un modèle doit être réentraîné lors de la réception de nouvelles données, et quelles sont les choix architecturaux et d'optimisation les mieux

adaptés pour ce contexte. Nous étudions ensuite l'utilisation de méthodes modulaires dans les tâches de traitement du langage naturel pour permettre une généralisation efficace en termes de données, tout en restant très efficace en termes de paramètres lors du transfert à de nouvelles tâches. En résumé, cette thèse contribue aux efforts en cours pour développer des modèles d'apprentissage automatique polyvalents et efficaces pour l'adaptation séquentielle, et souligne l'importance de traiter divers aspects de l'efficacité dans l'apprentissage continu.

Contributions to Original Knowledge

The following thesis makes several contributions towards understanding and improving the efficiency of continual learning methods. Specifically, we investigate several facets of efficiency in sequential knowledge acquisition, and make the following contributions :

1. A sample retrieval method for replay-based methods, designed to select datapoints most likely to be forgotten by the model (Chapter 4).

2. An online learned compression algorithm for non-stationary distributions, reducing the memory footprint of replay-based CL algorithms (Chapter 5).

3. An original analysis on the underlying causes of forgetting in class-incremental CL settings, and efficient solutions (Chapter 6).

4. A comprehensive study examining the effects of design decisions on efficiency and performance within sequential learning settings (Chapter 7).

5. A novel routing mechanism for modular networks, enabling highly parameter efficient transfer to new tasks (Chapter 8).

For all the contributions above, we release the code to reproduce our results. [1] [2] [3] [4] [5]

[1] https://github.com/optimass/Maximally_Interfered_Retrieval
[2] https://github.com/pclucas14/adaptive-quantization-modules
[3] https://github.com/pclucas14/aml
[4] https://github.com/facebookresearch/alma
[5] https://github.com/microsoft/mttl/

Contributions of Authors

- Chapters 1, 2 and 3, which include the introduction and the technical background material in this thesis, are written by me. I drew inspiration from several sources, including the theses of Rahaf Aljundi, Koustuv Sinha and Ryan Lowe.

- Chapter 4 is based on Aljundi et al. [2019d], which is a published conference paper in Advances in Neural Information Processing Systems (NeurIPS) 2019. The original idea was sparked by Rahaf Aljundi, and was further developed by me, Eugene Belilovsky and Massimo Caccia. Eugene and I iterated on the rehearsal method design, and I designed the hybrid autoencoder adaptation. I developed the implementation for our method and baselines for the rehearsal based experiments and the hybrid experiments, while Massimo handled the generative replay experiments. Rahaf and Eugene led the charge in writing the paper, with Massimo and I writing the experiments section. Laurent Charlin, Tinne Tuytelaars and Min Lin provided guidance on the direction and writing of the paper.

- Chapter 5 is based on Caccia et al. [2020a], which is a published conference paper at the International Conference on Machine Learning (ICML) 2020. I led the project: I developed and implemented the method, ran the experiments and contributed to the paper writing. Massimo Caccia and Eugene Belilovsky helped improve the experimental section, and contributed to the paper writing and presentation. Joelle Pineau provided consistent feedback and guidance throughout the several iterations of this project.

- Chapter 6 is based on Caccia et al. [2022a], which is a published conference paper at the International Conference on Learning Representations (ICLR) 2022. I led the project in the design of the experiments and the overall empirical investigation. I implemented the proposed method and baselines, and ran the experi-

ments with the help of Nader Asadi. Eugene Belilovsky, Rahaf Aljundi and Joelle Pineau helped brainstorm, iterate on the method, understand the results, and everyone contributed to the paper writing.

- Chapter 7 is based on Caccia et al. [2022c], which was published at the Conference on Lifelong Learning Agents (CoLLAs) 2022. The original idea for the empirical framework was proposed by Marc'Aurelio Ranzato. Ludovic Denoyers, Marc'Aurelio and I led the overall efforts in designing the study and interpreting the results. I led the charge in running the computer vision experiments, with Ludovic providing significant help on the codebase. Jing Xu and Myle Ott implemented and ran the NLP experiments of the paper. Marc'Aurelio wrote the first sections of the paper, I wrote the results section of the paper, and everyone else contributed with the writing and the several revisions of the paper.

- Chapter 8 was my internship project at MSR Montréal this past year, and was accepted in Advances in Neural Information Processing Systems (NeurIPS) 2023. The overall idea was proposed by me, and further refined across many brainstorming sessions with Alessandro Sordoni, Nicolas Le Roux and Edoardo Ponti. The implementation was a collaborative effort between me and Alessandro, as well as the overall design in experiments. Zhan Su helped with the final set of experiments. Matheus Pereira helped organize the codebase. Edoardo, Alessandro and I contributed to writing the paper, with help from Nicolas and Zhan.

Acknowledgements

I want to start by thanking Joelle, who has believed in me long before I believed in myself. Thank you for your continual support and mentorship, for never missing a meeting, for the opportunities put at my disposal, and most of all, for instilling in me the belief that one's well-being should always come first.

To Massimo, thank you for being the awesome big brother that you are, for giving me the extra nudge when I needed it, for sharing your research vision with me, and for showing me the human side of research.

I am truly grateful for the other people who have mentored me throughout my PhD. My gratitude goes to Laurent Charlin, Eugene Belilovsky and Rahaf Aljundi. It was truly a pleasure collaborating with you and learning the skills of the trade from you. To Marc'Aurelio Ranzato and Ludovic Denoyer, thank you for your guidance during my time at FAIR. To Alessandro Sordoni and Nicolas Le Roux, the past year at MSR was tremendously fun and enriching. Thank you for believing in me.

I have been very fortunate to meet wonderful colleagues and collaborators whom I now call friend over this last 6 years at McGill and MILA. This one goes out to Jad Kabbara, Philip Amortila, Pierre Thodoroff, Joey Bose, Thimothée Lesort, Koustuv Sinha, Andre Cianflone, Charles Onu, Clara Lacroce, Maxime Wabartha, Harsh Satija, Joshua Holla, Malik Altakrori, Sumana Basu, Jonathan Lebensol, and Thang Doan.

Sur une note plus personnelle, merci à mon entourage incroyable pour leur soutien constant au cours des dernières années. Merci pour votre sens de l'aventure et du ridicule, et de combiner la macaquerie à l'agréable. Merci à mes parents, Nicole et Paolo, pour m'avoir encouragé depuis le début, autant dans mes aventures académiques que sportives. Je suis incroyablement reconnaissant de tous

vos effort. Pour finir, merci à toi Geneviève, qui m'a supporté dans les hauts comme dans les bas. Merci de faire de moi une meilleure personne. I love you all.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

In today's rapidly evolving world, the importance of developing adaptable artificial intelligence (AI) systems cannot be overstated. Our environment is characterized by constant change, driven by technological advancements [Bubeck et al., 2023], economic shifts, and social transformations [Osofsky et al., 2020]. These changes necessitate the creation of AI systems that can effectively process and adapt to new information and circumstances, akin to the adaptability displayed by humans.

Current AI systems have made remarkable progress and demonstrated high performance on various well-defined tasks, including image recognition [Wortsman et al., 2022a] and generation [Saharia et al., 2022], natural language understanding [Chowdhery et al., 2022], and reinforcement learning [Reed et al., 2022]. However, despite their success in these specific applications, they struggle to achieve robust, continual adaptation in open-ended settings [Hadsell et al., 2020], where humans excel. Human adaptability allows for seamless adjustments to new situations, learning from novel experiences, and modifying their behavior to suit the ever-changing environment. This level of adaptability is a critical aspect that current AI systems, particularly deep neu-

ral networks (DNNs), need to develop in order to expand their utility in real-world applications that demand constant adaptation [Huyen, 2022a].

A major challenge faced by DNNs when learning sequentially is the phenomenon of *catastrophic forgetting* [French, 1999, McCloskey and Cohen, 1989]. In this scenario, DNNs quickly lose previously learned information when beginning to learn new tasks or data. This loss of knowledge significantly hampers their ability to learn and adapt over time, as they are unable to retain and build upon prior knowledge when confronted with new challenges [Kirkpatrick et al., 2017].

Currently, the de-facto solution in many real use cases is to train a new model from scratch on both the old and the new data, and to repeat this process whenever new data or information is made available [Huyen, 2022b, Shyam et al., 2019, Sener and Savarese, 2017]. This approach, however, is extremely inefficient as the dataset grows over time. Moreover, as the field is shifting towards larger foundation models [Bommasani et al., 2021] where a single training run incurs a substantial financial and environmental cost [Strubell et al., 2020, Schwartz et al., 2020], repeating this process is not a viable solution. To this end, enabling continual optimization of such large models can not only lead to models which can be efficiently adapted, it can also empower a wider portion of the research community to contribute to the active development of foundation models, as fewer resources are needed.

Moreover, Continual Learning can improve the robustness of currently deployed AI systems, by equipping them with the ability to learn from novel and unexpected situations. For instance, smart cars would be able to adapt to new traffic scenarios, from changing vehicle dynamics to new obstacles [Verwimp et al., 2022]. In AI-assisted healthcare, existing systems must adapt to new diseases or patient populations, and do so without regressing on previous standard of care. Given that sharing medical data is limited due to privacy concerns [Murdoch, 2021], simply re-training on the aggregated data may not be a feasible option.

Given these scenarios, it is evident that a solution should not only account for robustness against forgetting and scalability, but also provide a path for cumulative and iterative knowledge acquisition, a principle human learners employ naturally.

1.2 A Two Pronged Solution

Addressing the challenge of efficient sequential knowledge acquisition requires us to consider two integral facets of this complex issue. First, a learning episode should have *backward compatibility* [Shen et al., 2020], i.e. it should not inadvertently remove an ability which the model previously had. In other words, a viable solution must address the issue of catastrophic forgetting. A standard approach towards this goal is *experience replay* (ER) [French, 1999, Rolnick et al., 2018]. When learning a new task, data from previous tasks is interleaved with the new data. This incentivizes the model to adapt its internal knowledge to learn the new task, while keeping its current ability to handle previous tasks. This has been shown to be a reliable approach in many settings where the data distribution changes over time [Chaudhry et al., 2019b, Balaji et al., 2020]. However, replay-based solutions scale suboptimally as a function of the number of tasks. Indeed, since they must store data from previous learning iterations, the storage requirements must scale accordingly and this can be prohibitive, especially when dealing with high-dimensional data [Wang et al., 2022a]. Thus, how can we design new storage algorithms to minimize the memory footprint of replay? Moreover, as the model accumulates knowledge, the amount of replay steps to ensure knowledge preservation increases, along with its computational footprint [LESORT et al., 2023]. Therefore, how can we maximize the impact of each replay step given a fixed budget? In short, there remains significant progress to be made for ER to become an efficient long-term solution to learning without forgetting.

Second, Continual Learning can be seen through the lens of sequential forward

transfer. In other words, for an agent to perform optimally in a CL setting, it must become increasingly data and compute efficient over time. To do so, it must first achieve *transfer learning*, leveraging its existing capabilities that are relevant to a new task to aid learning, a behavior intrinsic to humans. In recent years, the go-to transfer learning paradigm is to first pretrain a model on large amounts of diverse data, before finetuning on a downstream task of interest [Brown et al., 2020a, Radford et al., 2019, Raffel et al., 2020]. Within this paradigm, how can we best adapt these pretrained models to incorporate new knowledge ? Given that standard fine-tuning may be vulnerable to forgetting [Ye et al., 2021], encapsulating new knowledge in external components, or *modular learning* [Ponti et al., 2023], is a good starting point. Then, how can we perform this transfer procedure iteratively to refine our pretrained model over time ? More generally, what challenges are *intrinsic* to learning in a sequential fashion, irrespective of a potential distribution shift from one learning episode to another, and what practical design choices can overcome these challenges?

1.3 Thesis Overview

This dissertation systematically addresses the research questions outlined earlier through five significant studies, divided across chapters 4 to 8. The initial chapters (4, 5, 6) focus on overcoming the challenges posed by rehearsal-based methods, whereas the latter sections (7, 8) delve into the nuances of transfer learning.

Chapter §4 investigates *which* samples should be selected for a given replay step [Aljundi et al., 2019d]. We first show that a training step on a given mini-batch does not cause uniform forgetting across all previous knowledge. We then propose a replay sample criterion selecting points that *maximally interfere* with the current mini-batch. We observe that this approach both limits forgetting while maximizing performance, making a better use of each replay step.

In Chapter §5, we tackle the storage bottleneck in rehearsal based methods, by learning a model which compresses data and stores the compressed encodings [Caccia et al., 2020a]. This is achieved by inserting a quantization bottleneck [Van Den Oord et al., 2017] in a standard autoencoder, which not only enables high compression rates, but enables us to regularize the encoded representations to maintain *backwards compatibility*, where the current decoder can still reconstruct representations from earlier encoder states.

Chapter §6 takes a step back and investigates *why* we observe a sudden decrease in performance in sequential classification settings where new classes are introduced over time [Caccia et al., 2022a]. We show that ER causes the newly added classes' representations to overlap significantly with the previous classes, leading to highly disruptive parameter updates. To counteract this, propose a new method which mitigates this issue by shielding the learned representations from drastic adaptation to accommodate new classes, effectively removing the sudden dip in performance at the introduction of new classes.

In Chapter §7, we shift our focus towards enabling forward transfer in settings where data arrive in large chunks over time. We first formalize this setting, Anytime Learning at Macroscale [Caccia et al., 2022c], primarily focusing on how best to allocate a compute budget over time. We empirically explore answers to critical questions within this framework, such as optimal wait times before training on newly arrived data, the most suitable model size and architecture, and methods for adapting the model's capacity throughout the learning process.

Finally, Chapter §8 explores how to maximize the efficiency of transfer to new tasks using pretrained language models and additional multi-task data [Caccia et al., 2022b]. We examine modular methods that simultaneously learn an inventory of modules and a routing function to select a subset of modules for each task. Our research suggests that such methods are effective due to their ability to mitigate interference and promote

transfer during multi-task training. Additionally, we propose a novel routing function to further enhance this optimization capability.

Chapter 2

Machine Learning Background

In this chapter, we provide a concise overview of the foundational concepts relevant to the research in this manuscript. We begin by introducing Machine Learning (ML) in the supervised learning setting, followed by Maximum Likelihood Estimation, and finally, artificial neural networks.

2.1 Supervised Learning

Supervised learning seeks to approximate an unknown target function $f^* : X \to Y$ using a labeled dataset D. This dataset consists of n input-output pairs, i.e. $D = \{(x_i, y_i)\}_{i=1}^n$. In this context, each input x_i belongs to the input space X and each output y_i to the output space Y. It is important to note that the samples in dataset D are drawn independently and identically distributed (i.i.d) from a joint probability distribution $P(X, Y)$. This critical assumption implies that each sample in the dataset is drawn independently of previous samples, from the same underlying distribution.

In the process of supervised learning, the goal is to learn a predictor $f(x; \theta)$, parameterized by θ, which will approximate the unknown target function f^*. The learning algorithm accomplishes this by minimizing a scalar loss function $\mathcal{L} : Y \times Y \to \mathbb{R}^+$, which

quantifies the discrepancy between the predicted output $f(x_i;\theta)$ and the true output y_i for each sample in the dataset.

Supervised learning primarily finds application in two settings: classification and regression. In classification, the target function $f^* : \mathbb{R}^d \to \{1, ..., k\}$ maps a real-valued vector $x \in X$ to a set of k discrete categories $y \in Y$. For instance, in medical diagnostics, the input x could be a patient's medical imaging, and the output y would be a categorical disease diagnosis [Cohen et al., 2020, Meedeniya et al., 2022]. On the other hand, in regression, the target function $f^* : \mathbb{R}^d \to \mathbb{R}$ maps a real-valued vector $x \in X$ to a real-valued output. An example of this is predicting house prices based on various features such as size, age, and location [Truong et al., 2020], where the output is a continuous value. Through these settings, supervised learning enables the generation of informed predictions and data-driven decision making.

2.2 Parameter Estimation

Parameter estimation in supervised learning involves finding a set of parameters θ that best explain the observed data. It is through this optimization procedure that learning occurs.

2.2.1 Maximum Likelihood Estimation

A fundamental approach to parameter estimation is Maximum Likelihood Estimation (MLE). MLE seeks to determine the parameters θ that maximize the likelihood of the observed data given the parameters, i.e. $L(\theta; D) = P(D|\theta)$. Formally, it is defined as

$$\theta_{MLE} = \arg\max_{\theta} \; P(D|\theta). \tag{2.1}$$

In practice, we typically maximize the log-likelihood for computational convenience. Moreover, given that standard practice in optimization is to minimize a loss function,

the MLE objective is reformulated as the negative log-likelihood

$$\theta_{MLE} = \arg\min_{\theta} -\log\left(P(D|\theta)\right). \tag{2.2}$$

2.2.2 Regularization

However, MLE can lead to overfitting if the model is too complex. To prevent this, regularization techniques [Tibshirani, 1996] are used. Regularization introduces a penalty term λ to the loss function to constrain the magnitude of the parameters, promoting model simplicity and generalization. A common example of this is l_2 regularization, also known as weight decay, which penalizes the l_2 norm of the parameters θ. The regularized MLE objective then becomes

$$\theta_{MLE} = \arg\min_{\theta} -\log\left(P(D|\theta)\right) + \lambda||\theta||_2. \tag{2.3}$$

2.2.3 Maximum a Posteriori Estimation

Another method of parameter estimation is Maximum a Posteriori (MAP) estimation. MAP introduces prior knowledge about the distribution of parameters through a prior distribution $P(\theta)$. It then optimizes the posterior probability $P(\theta|D)$, which we can relate to the likelihood term $P(D|\theta)$ via Bayes' Theorem,

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

Given that the marginal likelihood $P(D)$ does not depend on the parameters, the MAP estimate can be written as

$$\theta_{MAP} = \arg\min_{\theta} -\left[\log\left(P(D|\theta)\right) + \log P(\theta)\right]. \tag{2.5}$$

Interestingly, the MAP estimation can be viewed as a form of regularized MLE where the regularization term is determined by the choice of the prior distribution of the parameters. For example, choosing a Gaussian prior over θ renders equations 2.3 and 2.5 equivalent.

2.3 Neural Networks

2.3.1 Feed-forward Neural Networks

Feed-forward Neural Networks, also known as Multi-Layer Perceptrons (MLP) [Rosenblatt, 1958, Rumelhart et al., 1986, Goodfellow et al., 2016, Yan et al., 2015] are a family of highly expressive functions, which can capture non-linear relationships between X and Y. MLPs chain multiple fully-connected layers. Each layer first linearly transforms an input, then applies a differentiable non-linear activation function $a(\cdot)$:

$$h^l = a(h^{l-1} \cdot W^l + b^l), \tag{2.6}$$

where h^l is the output of layer l, with $W^l \in \mathbb{R}^{d_{in} \times d_{out}}$, $b^l \in \mathbb{R}^{d_{out}}$ as learnable parameters. As its name suggests, each layer computes its hidden output then forwards it to the next layer, until the last layer is reached. Given an MLP with L layers, we can express the full function as

$$f(x; \theta) = (h^L \circ h^{L-1} \circ ... \circ h^1)(x), \tag{2.7}$$

where $\theta = \{(W^i, b^i)\}_{i=1}^L$.

The choice of activation function $a(\cdot)$ for the last layer will depend on the setting at hand. For classification tasks, the softmax activation is the de-facto approach to turn unnormalized logits into a proper probability distribution, with $\text{softmax}(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$. In other words, $\text{softmax}(x)_i$ represents the probability that the input belongs to i-th class. In regression settings, if the target values are unnormalized real-values outcomes, one

can omit the activation function. For intermediate, or *hidden* layers, several activation functions have been proposed over the years, including the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and the rectified linear unit $\text{ReLU}(x) = \max(x, 0)$ [Glorot et al., 2011].

2.3.2 Architectures

In the realm of deep learning, various architectures have been developed to handle different types of data and tasks. These architectures reflect the need to encode different forms of structural priors and inductive biases into the model design to effectively learn from data.

Convolutional Neural Networks (CNNs) [Krizhevsky et al., 2017, He et al., 2016] are specialized for processing grid-like data such as images. Such objects introduce a *height* and *width* dimension, with each point on this grid containing a vector in $\mathbb{R}^{d_{in}}$. Given as input a tensor of shape (H, W, D), CNNs share parameters across different regions of the (H, W) axis. In other words, for a given layer, each image patch is processed by the same feature extractor, and the outputs are combined such that the 2D structure is preserved. This allows the network to reuse the same weights to detect similar features across different regions of the input, giving layers that are more parameter efficient than their fully connected counterpart, and equivariant to translations. Formally, for a convolutional layer with input h^{l-1} , we can express its output at pixel location $[i, j]$ as

$$h^l[i, j] = a\left(\sum_m \sum_n h^{l-1}_{[i-m,j-n]} \cdot W^l_{[m,n]} + b^l\right), \tag{2.8}$$

where $b^l \in \mathbb{R}^{d_{out}}$ and $W^l_{[m,n]} \in \mathbb{R}^{d_{in} \times d_{out}}$, and (m, n) the patch size. As with fully-connected layers, convolutional layers can also be composed one after the other to build a full network. CNNs are a good example on how Deep NNs build hierarchical representations, where early layers respond to basic edge patterns, while intermediate and end layers can detect object parts and objects respectively [Katole et al., 2015].

Recurrent Neural Networks (RNNs) [Elman, 1990, Jordan, 1986] [Hochreiter and Schmidhuber, 1997] are particularly well-suited to sequence-like data such as natural language, where sentences are encoded as sequences of words. RNNs have an inherent loop structure that enables them to process one element of the sequence at a time while maintaining a hidden state that encapsulates the information from previous elements. This approach enables RNNs to process input sequences of varying length, unlike previous architectures. Similar to how CNNs share weights across different image patches, in RNNs this sharing is enabled across different tokens in the sequence. The forward propagation step for RNNs can be instantiated in many ways [Goodfellow et al., 2016]. In one such instantiation for a recurrent layer with $h_{(t-1)}$ denoting the previous hidden state, omitting the superscript l we can express its hidden state $h_{(t)}$ as

$$h_{(t)} = a\left(h_{(t-1)} \cdot W^h + x_{(t)} \cdot W^x + b\right), \tag{2.9}$$

where $b \in \mathbb{R}^d$, $W^h \in \mathbb{R}^{d \times d}$ and $W^x \in \mathbb{R}^{d_{in} \times d}$ are the bias and weight matrices, respectively, and a is an activation function. The power of RNNs lies in their ability to learn temporal dependencies and build up complex temporal structures over time.

Autoencoders [LeCun, 1987, Hinton and Zemel, 1993] are unsupervised learning models that aim to learn a compact, efficient representation of the input data, typically for the purpose of dimensionality reduction or denoising. They consist of two parts: an encoder $\texttt{enc}(\cdot)$, which transforms the input into a hidden representation, and a decoder $\texttt{dec}(\cdot)$, which reconstructs the input from this representation. The autoencoder tries to minimize the reconstruction error:

$$\mathcal{L}(x, \texttt{dec}(\texttt{enc}(x))) = ||x - \texttt{dec}(\texttt{enc}(x))||_2^2. \tag{2.10}$$

In denoising autoencoders [Vincent et al., 2008], the input is intentionally corrupted with noise before being fed into the autoencoder, encouraging the model to learn to reconstruct the original, noise-free data. These models are effective for feature extraction and data compression tasks.

2.3.3 Training via backpropagation

The backpropagation algorithm [Rumelhart et al., 1985] is a critical component in training neural networks, allowing for efficient computation of gradients of the loss function $\mathcal{L}$ with respect to the model's parameters θ. It exploits the structure of the computational graph of a neural network and the chain rule from calculus to propagate errors backward through the layers of the network. For each forward computation $h^l = a(h^{l-1} \cdot W^l + b^l)$ in a given layer l, we first compute the gradient w.r.t h^l. Then taking the gradient of the output h^l w.r.t to the layer parameters $\theta^l = (W^l, b^l)$, and applying the chain rule we obtain the desired derivative of the loss w.r.t to the model parameters:

$$\frac{\partial \mathcal{L}}{\partial h^l} = \frac{\partial \mathcal{L}}{\partial h^{l+1}} \cdot \frac{\partial h^{l+1}}{\partial h^l}, \qquad \frac{\partial \mathcal{L}}{\partial \theta^l} = \frac{\partial \mathcal{L}}{\partial h^l} \cdot \frac{\partial h^l}{\partial \theta^l}. \tag{2.11}$$

The backward pass is executed by unrolling the computation graph from the last layer to the first one, $l = L, ..., 1$, executing in a sequential fashion Eq. 2.11. Backpropagation efficiently computes these derivatives by caching intermediate outputs in the forward pass and using them in the computation of gradients (backward pass). This reuse of computed values greatly enhances computational efficiency, making backpropagation a practical method for training deep neural networks. Once the gradients have been computed, the model parameters are updated by taking a step in the direction of the negative gradient, the step size determined by the learning rate α

$$\theta^l \leftarrow \theta^l - \alpha \frac{\partial \mathcal{L}}{\partial \theta^l}. \tag{2.12}$$

Several variants of this gradient descent approach are used in practice, namely the using of a momentum term [Polyak, 1964, Rumelhart et al., 1985], and per-parameter learning rates [Kingma and Ba, 2015] for better convergence.

Chapter 3

Continual Learning

In the previous Chapter, we discussed how the i.i.d. assumption plays a crucial role in deriving the Maximum Likelihood Estimation (MLE) estimator. It provides a theoretical basis for MLE, ensuring that the learned model captures the underlying data distribution effectively. However, in practical scenarios, this assumption does not always hold. Real-world data often exhibit non-stationary distributions, where changes in the environment lead to new patterns or trends over time. For instance, in the case of an object detector, we would like the model to adapt to and recognize novel items as they are introduced. Similarly, a streaming platform's recommender system should be able to accommodate users' evolving preferences and consider new releases in its predictions. In both examples, the distribution of data changes dynamically, violating the i.i.d. assumption. Continual Learning emerges as a promising solution for such situations, as it addresses the challenge of enabling models to adapt efficiently and effectively to the changing data distribution. By relaxing the i.i.d. assumption and allowing for distribution shifts over time, Continual Learning empowers models to evolve and refine their knowledge in accordance with the dynamic nature of real-world data.

3.1 Problem Formulation

Continual Learning considers a never-ending stream of data. At each time step t, the learner receives a data sample (x_t, y_t) or a set of such data samples, where $x_t \in X$ and $y_t \in Y$. It is important to note that these samples are drawn non-i.i.d. from a target distribution $P_t(X, Y)$, which itself is time-dependent. As the environment changes over time, the underlying distribution $P_t(X, Y)$ evolves, violating the i.i.d. assumption. The objective of CL is to learn a predictor $f(x; \theta_t)$ parameterized by θ_t, that minimizes the loss on the current data (x_t, y_t). However, the optimization procedure must be constrained to ensure the retention of previously learned knowledge, as the model adapts to new data samples. Formally, the CL objective can be written as

$$\arg\min_{\theta,\xi} \; \mathcal{L}(f(x_t;\theta), y_t) + \sum_i \xi_i \tag{3.1}$$

$$\text{s.t.} \quad \mathcal{L}(f(x_i;\theta), y_i) \leq \mathcal{L}(f(x_i;\theta_{(t-1)}), y_i) + \xi_i \tag{3.2}$$

$$\xi_i \geq 0; \; \forall i \in \{1, ..., t-1\}.$$

In equation 3.2, the slack variables $\xi = \{\xi_1, ..., \xi_t\}$ provide a flexible margin for constraint satisfaction. In other words, this formulation can model settings where some forgetting on a subset of data is permitted to enable a strong performance boost over other, potentially more relevant parts of the data distribution.

3.1.1 Continual Learning Desiderata

In this section, we list multiple key characteristics that are required for a Continual Learner to solve equation 3.1 in a tractable manner.

1. Constant Memory In a real-world setting, data arrive continuously, and the volume of data may grow indefinitely over time. It is neither practical nor efficient to store all the incoming data indefinitely. This desideratum ensures that the memory footprint of the learning system stays fixed, irrespective of the amount of seen data. Optimally, the learner should operate without storing *any* past data [De Lange et al., 2021].

2. Bounded Compute Another essential requirement for Continual Learning is the ability to operate under bounded compute constraints. As the model encounters new information and tasks over time, its computation footprint to learn a new task should not grow over time. For example, retraining from scratch on all the data accumulated does not satisfy this constraint. In general, operating under bounded compute constraints ensures that the learning system remains practical in real-world applications, where computational resources are often limited.

3. Online or Anytime Learning Continual Learning should enable models to learn and adapt to new information on-the-fly, without the need for periodic offline retraining. Online or Anytime Learning allows models to incorporate new information and update their knowledge as soon as it becomes available. This ability is particularly crucial in dynamic environments where the data distribution changes rapidly, and the model's predictions must remain valid and up-to-date at all times. Online or Anytime Learning ensures that the Continual Learning system remains responsive and adaptive in the face of evolving data patterns and challenges.

4. Forward and Backward Transfer A key aspect of Continual Learning is the ability to transfer knowledge from previously learned tasks to new tasks (forward transfer) and vice versa (backward transfer). Forward transfer enables models to leverage previously acquired knowledge to learn new tasks more efficiently, while backward transfer allows models to improve their performance on previous tasks as they encounter new

information. This bidirectional transfer of knowledge is a crucial component of human learning and is essential for artificial learning systems to achieve robust adaptation to new tasks and challenges.

5. Task Agnostic In many real-world applications, task identifiers or explicit task boundaries may not be available. Continual Learning algorithms should be designed to operate in a task-agnostic manner, meaning that they should not rely on task identifiers to make valid predictions or guide learning. Instead, models should be able to autonomously identify and adapt to changes in the data distribution, without the need for explicit task information. This task-agnostic approach ensures that Continual Learning algorithms can be applied effectively in diverse real-world scenarios, where task boundaries or identifiers might not be readily accessible.

3.2 Problem Instantiations

In practice, solving the general Continual Learning problem, while respecting all the desiderata stated above, is very challenging. Indeed, given that each desideratum adds a level of complexity to the solution, there exists many CL settings where some of the constraints above are relaxed. Here, we present some of the CL settings which will be discussed throughout this thesis. For a more in-depth categorization of CL settings, we refer the reader to van de Ven et al. [2022], Normandin et al. [2021].

3.2.1 Task-Incremental Learning (TIL)

In this simplified setting [Kirkpatrick et al., 2017], we assume that there exists a notion of task $\mathcal{T}$, each with a *fixed* target distribution $P_{\mathcal{T}}(X, Y)$. The learner then obtains access to each task sequentially, one task at a time. Crucially, while the learner is training on data from task $\mathcal{T}$, it has access to i.i.d samples from $P_{\mathcal{T}}(X, Y)$. Typically, when

transitioning to a new task $\mathcal{G}$, the learner relinquishes access to data from $\mathcal{T}$, unless explicitly leveraging an external bounded memory to store seen samples of said task.

Online TIL In Online Task-Incremental Learning settings [Lopez-Paz and Ranzato, 2017], the agent again learns online from a stream of data. Crucially, datapoints from two consecutive timesteps (x_t, y_t) and (x_{t+1}, y_{t+1}) are i.i.d samples from the same underlying task distribution $P_{\mathcal{T}}(X, Y)$, unless a task switch has occured. When task switch occurs, the new task $\mathcal{G}$ and its underlying distribution $P_{\mathcal{G}}(X, Y)$ will yield consecutive i.i.d samples in the stream, until the next task switch. The data stream can be seen as being locally stationary. During learning, the agent does not know when the next task switch will occur. Unless explicitly stored in an external memory, the online nature of this setting prohibits data from previous timesteps to be accessed (even ones from the same task).

Offline TIL In Offline Task-Incremental Learning [Rebuffi et al., 2017], at every learning step the learner is provided with a task specific dataset $D_{\mathcal{T}} = \{(x_i, y_i)\}_{i=1}^{N_{\mathcal{T}}}$. The learner can then perform offline updates to its parameters, incorporating information contained in $D_{\mathcal{T}}$ with potentially multiple passes over the dataset. In such a setting, the agent is always task-aware during training, and does not face the challenges related to online optimization.

In this thesis, for both settings, we typically assume that the learner is task agnostic at evaluation time. In other words, when querying the model, no task information is given to the agent. For example, if every task is a binary classification problem, the learner trained on T tasks must perform $(T \times 2)$-way classification at test time.

3.2.2 Online Class-Incremental Learning (CIL)

In this work, we mainly focus on the **online class-incremental** setting [Aljundi et al., 2019c], an instantiation of online TIL. Here, the learner must classify images as belonging to an increasing set of candidate classes. That is, each task is a $K-$way classification task, and by construction (x, y) pairs in task $\mathcal{T}$ contain labels that are distinct from seen classes in previous tasks. In other words, new classes are incrementally added to the set of candidate classes. Depending on the setting, the task boundary may or may not be given explicitly to the learner during training. This setting is depicted in figure 3.1.

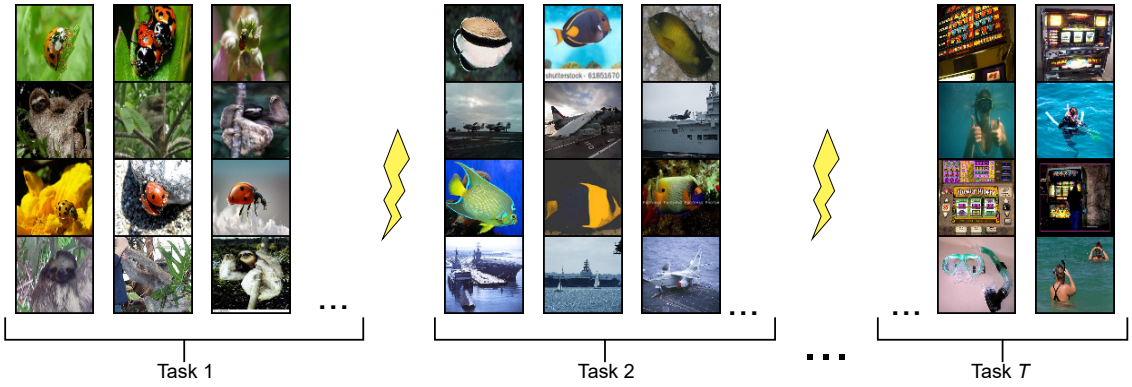

Figure 3.1 Online Class Incremental Setup. In this example, the learner is presented two new classes at every task. Task boundaries are denoted by lightning bolts. Each column of data represents the incoming batch received by the learner at every timestep.

3.2.3 Terminology and Evaluation

In the supervised setting, a continual learner should perform well across all tasks seen so far. Specifically, we want to maximize the Average Accuracy [Lopez-Paz and Ranzato, 2017, Chaudhry et al.] across all tasks seen in the data stream so far. For each task $\mathcal{T}$, a held-out set of samples $D_{\mathcal{T}}^{(eval)}$ is used for evaluation purposes. Let $a_{k,j} \in [0, 1]$ denote the accuracy measured on $D_j^{(eval)}$ after training on task k (it follows that $k \geq j$). We denote the average accuracy after k tasks by

$$A_k = \frac{1}{k} \sum_{t=1}^{k} a_{k,t} \ . \tag{3.3}$$

We also keep track of how much the model forgets at it learns new tasks. We define the **forgetting** [Chaudhry et al.] for a given task as the difference between the best performance obtained over time by the model and the final performance on said task. After training on k tasks, forgetting is defined as

$$F_k = \frac{1}{k-1} \sum_{t=1}^{k-1} f_j^k, \tag{3.4}$$

where f_j^k denotes the forgetting on the j-th task occuring after training on task k has completed. This value is computed as

$$f_j^k = \max_{l \in \{1,\dots,k-1\}} a_{l,j} - a_{k,j}. \tag{3.5}$$

Naturally, a good agent should obtain both high accuracy and low forgetting, however there is often a trade-off between these two metrics. We typically report the final average accuracy and forgetting after the last task (T), namely A_T and F_T.

3.3 Continual Learning Approaches

Early work in the field connectionism [McCloskey and Cohen, 1989, French, 1999] had already observed catastrophic forgetting when using small artificial neural networks in memorization tasks. Several prior works in the literature attributed representation overlap [French, 1994, 1999, Kruschke, 1992, 1993, Rumelhart, 1992] as the key factor behind catastrophic interference. Other methods have been proposed that explicitly minimize overlap by encouraging the learning of sparser, orthogonal representations. For example, methods such as Activation Sharpening [French, 1991] would push the

top-k hidden units towards 1, and the remaining to 0. In the early years of CL research was also studied the use of replaying patterns seen in the past to mitigate forgetting. Originally proposed in Ratcliff [1990], the use of recency rehearsal, where the last p patterns were kept for replay, was shown to partly remedy forgetting. Explored in more depth in Robins [1995], it was shown that selecting a more representative set of replay samples (via random selection of past patterns) was more effective, and that one could also replay all previous samples to fully mitigate any loss in performance, albeit at an increased computational cost. Interestingly, the concept of pseudo-rehearsal, where random data was generated and labelled with the model itself, before being used as valid rehearsal data was proposed. In a similar vein, it was also shown that labelling new data with networks trained on previous tasks [Silver and Mercer, 2002] could create useful virtual samples for replay. These works can be seen through the lens of distillation [Hinton et al., 2015], and would inspire more recent methods [Li and Hoiem, 2018]. However, early works in the field mainly tackled very small and synthetic sequences of memorization tasks, not focusing on generalization to new samples from the same distribution.

After the emergence of Deep Neural Networks and the significant increase in computation afforded by GPUs, the study of catastrophic forgetting in DNNs when subjected to sequential optimization surfaced again as a prominent issue. It was shown in Goodfellow et al. [2013] that the choice of activation function and regularization techniques used during training can impact forgetting in sequential learning, and subsequently that architecture [Mirzadeh et al., 2022] and optimization [Mirzadeh et al., 2020] choices also come into play.

In recent years, many methods have been proposed for Continual Learning. They can be broadly grouped in three categories, each approaching the problem of how to store, re-use and consolidate old knowledge with new one. Firstly, *replay* based methods store externally previously seen patterns which are then interleaved alongside new

ones to effectively remind the network and ensure knowledge preservation. Second, *regularization* based methods enforce constraints (in parameter of functional space) discouraging the model to overwrite useful parts of the network when consolidating new knowledge. Lastly, *architectural* based approaches allocate different subsets of parameters, or modules, to different tasks to ensure that no overwriting of important information occurs. We refer the reader to De Lange et al. [2021] for a more detailed overview of CL methods.

3.3.1 Replay Based Methods

Algorithm 1 EXPERIENCE REPLAY (ER)

Input: Learning rate α, Number of iterations N_{iter}

Initialize: Memory $\mathcal{M}$; Parameters θ **do**
  Receive $\mathbf{X}^{in}, \mathbf{Y}^{in}$ `// Receive from Stream`
  for $n \in 1 \dots N_{iter}$ **do**
    $\mathbf{X}^{bf}, \mathbf{Y}^{bf} \sim \text{SAMPLE}(\mathcal{M})$ `// Sample from Buffer`
    $\mathcal{L}_{inc} = \mathcal{L}(\ f_\theta(\mathbf{X}^{in}), \mathbf{Y}^{in}\ )$ `// Compute incoming loss`
    $\mathcal{L}_{bf} \ = \mathcal{L}(\ f_\theta(\mathbf{X}^{bf}), \mathbf{Y}^{bf}\ )$ `// Compute replay loss`
    $\theta \ := SGD(\nabla(\mathcal{L}_{inc} + \mathcal{L}_{bf}), \theta, \alpha)$ `// Update model parameters`
  UPDATEMEMORY$(\mathcal{M}, \mathbf{X}^{in})$ `// Update Buffer Memory`
while *The stream has not ended*

Replay has been a consistent and effective approach in various continual learning settings and is extensively used in Reinforcement Learning to decorrelate samples within a trajectory [Mnih et al., 2013, Schaul et al., 2015]. Numerous forms of replay exist, with Algorithm 1 representing the general form in an online setting. Over the years, several variants have been proposed, leveraging the stored data for different purposes. In

classification settings, Rebuffi et al. [2017] use stored samples to compute average embeddings for each class and make predictions through a nearest-class-mean approach. Alternatively, stored samples can be used for distillation, ensuring the new model's output on old samples remains close to previous predictions [Buzzega et al., 2020, Boschini et al., 2022, Castro et al., 2018, Douillard et al., 2020]. In class-incremental settings, prototypes in the last linear layer are known to be biased towards more recent classes, and methods have been proposed to alleviate the issue [Hou et al., 2019, Ahn et al., 2021]. Moreover, Replay with small buffers can lead to overfitting, and explicit regularization can help counter this [Bonicelli et al., 2022]. To address the computational and memory cost of rehearsal, instead of using raw samples, replay can also be performed in feature space [Pellegrini et al., 2020, Hayes et al., 2020, Ostapenko et al., 2022]. Replay can also be performed with generated data. Although pseudo-rehearsal with random vectors and pseudo labels does not work with DNNs, generative models can be used instead for continual training [Atkinson et al., 2018, Shin et al., 2017, Van de Ven et al., 2020]. While current generative models can produce highly realistic data [Saharia et al., 2022], training them in a continual setting remains challenging [Lesort et al., 2019a]. To increase storage efficiency, compressed samples can be obtained via an autoencoder [Riemer et al., 2019] or via standard compression algorithms [Wang et al., 2022a]. Regarding buffer management techniques related to sample selection, a de-facto approach is Reservoir Sampling [Vitter, 1985, Rolnick et al., 2018, Riemer et al., 2018], which ensures a representative subset of the full data stream is kept in memory. Sample selection can also be formulated as a constrained optimization procedure for promoting diversity [Aljundi et al., 2019c], which can be useful in settings where the data stream is unbalanced. When additional task or class information is available, it can be leveraged for more balanced sample selection [Yoon et al., 2022, Shim et al., 2021, Rebuffi et al., 2017]. Lastly, samples in the memory can also be used to perform constrained optimization, where gradients from different tasks do not interfere with

each other. In Gradient Episodic Memory (GEM) Lopez-Paz and Ranzato [2017], the authors propose to formulate this constraint by enforcing that the dot product of task gradients to be non-negative, which is achieved by projecting the estimated gradient direction of the new tasks in the feasible region determined by previous task gradients. AGEM [Chaudhry et al.] proposes to relax this constraint, projecting the new task gradient on the average gradient direction determined over a randomly sampled subset of previous tasks. In summary, replay based approaches are the most robust and best performing methods among the three categories, but performing the rehearsal step increases the computational and memory cost.

3.3.2 Regularization based Methods

Regularization-based methods for Continual Learning can be broadly categorized into two groups: data-focused methods and prior-focused methods.

Prior-focused methods focus on constraining the model's parameter updates to preserve important parameters from previous tasks. This is done via a Bayesian approach, where we aim to learn the posterior probability of our parameters given the data. Let us consider a TIL scenario with two tasks and their respective datasets D_1 and D_2. When presented with the second task, the Bayesian learning objective is to find parameters θ maximizing the log posterior $\log p(\theta|D_1, D_2)$. Prior-based methods rely on Bayes' Rule to rewrite this term without requiring access to the previous dataset :

$$\begin{aligned}
\log p(\theta|D_1, D_2) &= \log p(D_2, D_1, \theta) - \log p(D_1, D_2) \\
&= \log p(D_2|\theta, D_1) + \log p(\theta|D_1) + \log p(D_1) - \log p(D_1, D_2) \\
&= \log p(D_2|\theta) + \log p(\theta|D_1) - \log p(D_2|D_1)),
\end{aligned}$$

with the last line leveraging the conditional independence of D_1 and D_2 given θ. Here, the first term is the log-likelihood of the new task, i.e. the standard MLE objective computed on D_2. The last term does not depend on θ and can be ignored during optimization, and can be dropped. Importantly, the middle term $p(\theta|D_1)$ is the posterior in which all the information from the first task is encoded. Prior-focused methods leverage the posterior of the first task as a *prior* for the second task, which constrains the optimization procedure from erasing knowledge from Task 1 by discouraging parameters that are unlikely under the prior.

Unfortunately, estimating the true posterior distribution is intractable, therefore approximations are used instead. Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017] uses the Laplace approximation [MacKay, 1992], which assumes the posterior follows a Gaussian distribution centered on θ_1 (the MLE estimate obtained by training on D_1) with diagonal covariance. This results in a quadratic penalty on the change of model parameters, with a weighting proportional to the Empirical Fisher Information Matrix (EFIM) of the previously learned tasks. Synaptic Intelligence (SI) [Zenke et al., 2017], interprets the prior term in EWC as a regularizer, and proposes a new one which approximates the contribution of each parameter to the overall loss. They then use this information to regularize the updates, preventing critical parameters from being changed drastically. Riemannian Walk [Chaudhry et al., 2018], draws keys concepts from EWC and SI, and combines the use of the EFIM, as well as a per-parameter importance score to constrain the optimization process, resulting in a more stable learning trajectory. Similarly, Memory Aware Synapses [Aljundi et al.] proposes a way to estimate the importance of each parameter for the past tasks in a fully unsupervised and online manner. To ensure that the diagonal EFIM approximation of the posterior in EWC is closer to the true posterior, Liu et al. [2018] learn a reparameterization of the parameters which minimizes this approximation error. Lastly, Variational Continual Learning (VCL) [Nguyen et al., 2018] employs variational inference [Kingma and

Welling, 2014] to approximate the posterior distribution of model parameters given the data and regularizing the model by minimizing the Kullback-Leibler (KL) divergence between the prior and posterior distributions.

Data-focused methods all leverage a form of knowledge distillation, typically between a previous version of the model and the current one being optimized, effectively grounding the model close to previous task solutions. One such example is Learning without Forgetting (LwF) [Li and Hoiem, 2017], which uses knowledge distillation, ensuring that the current model's output on previous tasks remains close to its previous predictions while learning the new task. Less-forgetting Learning in Deep Neural Networks [Jung et al., 2016] matches features at the network's penultimate layer between old and current model via an l_2 loss. Encoder Based Lifelong Learning [Rannen et al., 2017] trains task-specific autoencoders on the hidden features of a shared classifier. These autoencoders are then used to regularize training on a new task, via an auxiliary loss preventing the reconstructed features from changing significantly. Finally, Class-incremental learning via deep model consolidation [Zhang et al., 2020] uses a two-stage learning process that first learns a separate model on the new task, and then consolidates their knowledge using distillation, yielding back a single model.

To summarize, regularization based approaches are a solution to Continual Learning without storing past data, and prior based methods are well grounded in Bayesian Theory. While this family of methods might be effective in the task incremental setting with a small number of disjoint tasks, this family often shows poor performance when tasks are similar and training models are faced with long sequences as shown in Farquhar and Gal [2018]. In such settings, replay-based methods are preferred.

3.3.3 Architecture based Methods

This family of methods leverages the task-identifier during training to explicitly allocate subsets of parameters which are only modified during a single task. In other words, when a new task is encountered, previous parameters are kept frozen, while additional ones are optimized on the given task. At inference, these methods require either the task identifier of the data given to the model, or an explicit mechanism to infer the task id from the data. Such methods are by design immune to catastrophic forgetting, as the task-specific model can be retrieved exactly, by discarding parameters specific to other tasks. Progressive Neural Networks [Rusu et al., 2016] instantiates a new set of weights for each task, which are connected to the - frozen - weights of previous tasks belonging to the same layer, similar to Xu and Zhu [2018]. ExpertGate [Aljundi et al., 2017] trains a disjoint expert and an autoencoder for each task. At inference time, the autoencoders are used as a routing mechanism, weighting the output of each expert proportionally to their ability to reconstruct the input data. [Gaya et al., 2023] learns a set of anchor weights for each layer and learns task-specific mixture coefficients used to average said weights. When a new task is given, previous anchors are frozen, and a new anchor is dynamically added if a weighted average of existing anchors performs suboptimally. [Wortsman et al., 2020] learns a binary mask activating only a subset of weights. The authors also propose a task inference mechanism, which performs gradient-based search over possible task-specific masks. Other approaches operate under a fixed architecture, and rather than adding new parameters for each, a parameter (or activation) subset of the existing model is identified as belonging to a specific task, and is only updated during said task [Mallya and Lazebnik, 2018, Fernando et al., 2017, Serra et al., 2018]. Overall, these methods shine in settings where task inference is achievable. The main drawback of these methods is that the number of parameters usually increases when learning new tasks, which can be unfeasible in some cases.

3.4 Relation To Other Fields

Continual Learning, while a distinct field, shares many connections and similarities with other areas of research in machine learning. In this section, we briefly discuss the relationships between Continual Learning and other related fields such as Transfer Learning, Multi-Task Learning, Meta-Learning, and Online Learning.

3.4.1 Transfer Learning

Transfer Learning [Zhuang et al., 2020, Pan and Yang, 2010, Weiss et al., 2016] is the process of leveraging knowledge acquired from a source task or domain to improve learning in a related target task or domain. The core idea in Transfer Learning is that, instead of learning from scratch, models can benefit from previously acquired knowledge to achieve better performance or learn more efficiently. Continual Learning shares similarities with Transfer Learning in that both fields are concerned with learning from multiple tasks or domains. However, the key difference lies in the sequential nature of learning in Continual Learning, where models must adapt to new tasks over time while retaining knowledge of previous tasks, whereas Transfer Learning typically focuses on transferring knowledge from a single source task to a single target task, irrespective of a potential performance decrease on the source task. In recent years, the "pretrain and finetune" paradigm [Brown et al., 2020a, Radford et al., 2019, Raffel et al., 2020] has emerged as a powerful transfer learning approach, where a model is first pretrained without supervision on large amounts of data, before being finetuned on a downstream task of interest. A similar approach is also seen in Continual Learning, where the initial model prior to sequence learning is a foundation model [Ostapenko et al., 2022].

3.4.2 Multi-Task Learning

Multi-Task Learning (MTL) [Caruana, 1997, Ruder, 2017] is a learning paradigm where models are trained on multiple tasks simultaneously, with the goal of improving generalization performance on each task. MTL is based on the assumption that solving multiple tasks jointly can lead to the discovery of shared representations or structures, resulting in better performance compared to learning each task independently. Continual Learning and MTL share the common objective of learning from multiple tasks. However, in Continual Learning, tasks are encountered sequentially, whereas MTL focuses on learning tasks concurrently. Moreover, Continual Learning faces the additional challenge of mitigating catastrophic forgetting as the model encounters new tasks, which is not a concern in MTL.

3.4.3 Meta-Learning

Meta-Learning, or "learning to learn" [Thrun and Pratt, 1998, Finn et al., 2017], focuses on designing learning algorithms that can adapt quickly to new tasks by leveraging prior experience or knowledge. The primary aim of Meta-Learning is to enable models to learn new tasks or concepts with minimal supervision and few examples. While Continual Learning and Meta-Learning both deal with learning from multiple tasks or domains, Meta-Learning performs offline training, assuming that all training data is available at once. Moreover, Meta-Learning often emphasizes the ability to rapidly adapt to new tasks irrespective of performance degradation on earlier tasks, while Continual Learning is concerned with the long-term adaptation and retention of knowledge across a sequence of tasks. Lastly, combining key concepts of Meta-Learning and Continual Learning is an active area of research and has shown promising results [Javed and White, 2019, Beaulieu et al., 2020, He et al., 2016].

3.4.4 Online Learning

In Online Learning [Cesa-Bianchi and Lugosi, 2006, Hazan et al., 2016], models are updated incrementally as data samples become available, typically in a streaming fashion. That is, the learner must make predictions or decisions for each incoming data sample before receiving feedback. Continual Learning is closely related to Online Learning, as both fields address the challenge of learning from a stream of data. However, Continual Learning focuses on the adaptation to non-stationary data distributions and the retention of knowledge across tasks, whereas Online Learning primarily deals with the optimization of learning in a streaming data setting.

Chapter 4

Replay Sample Selection through Maximally Interfered Retrieval

So far, we have established that replay based methods are a robust approach to continual learning, outperforming regularization based approaches. This gain, however, comes at an increased cost in computation, which is required to perform the rehearsal step. In this chapter, we investigate whether current standard practices during the replay step are optimal. Specifically, we direct our attention towards answering the question of *what samples should be replayed from the previous history when new samples are received*. Indeed, the standard approach of randomly selecting rehearsal data independently of the current task being learned [Chaudhry et al., 2019b, Aljundi et al., 2019c], implicitly assumes that all prior knowledge is impacted uniformly when acquiring new knowledge. Intuitively, learning a new task should result in limited interference with a previous task if the two map to very orthogonal hidden representations under the model. Thus, replaying data from the previous task may not be necessary. Alternatively, if a new task maps to similar features as a previous task, then the model's internal knowledge of this older task is more likely to be impacted if the current shared representation changes significantly to accommodate the new task. In this situation,

replaying data from the previous task is crucial to mitigate forgetting.

The contribution proposed in this chapter leverages this intuition, and formalizes into an objective that is used to select samples for rehearsal. The proposed approach, Maximally Interfered Retrieval (MIR), favors rehearsal data whose prediction under the model will be the most negatively impacted from training on the incoming data. MIR also takes some motivation from neuroscience where replay of previous memories is hypothesized to be present in the mammalian brain [McClelland, 1998, Rolnick et al., 2018], but likely not random. For example it is hypothesized in Honey et al. [2017], Long NM that similar mechanisms might occur to accommodate recent events while preserving old memories. Our proposed approach works with both standard replay and rehearsal with generative models (§ 4.1), enabling gradient based search of these interfered points in the latter. We evaluate our approach in an online continual learning setting, where the data is seen only once and is not iid. We show that MIR not only reduces forgetting, but also leads to consistent gains in accuracy (§ 4.2).

4.1 Methods

We consider a (potentially infinite) stream of data where at each time step t, the system receives a new set of samples $\boldsymbol{X}_t, \boldsymbol{Y}_t$ drawn non i.i.d from a current distribution D_t that could itself experience sudden changes corresponding to task switching from D_t to D_{t+1}.

We aim to learn a classifier f parameterized by θ that minimizes a predefined loss $\mathcal{L}$ on new sample(s) from the data stream without interfering, or increasing the loss, on previously observed samples. One way to encourage this is by performing updates on old samples from a stored history, or from a generative model trained on the previous data. The principle idea of our proposal is that instead of using randomly selected or generated samples from the previous history [Chaudhry et al., Shin et al., 2017], we

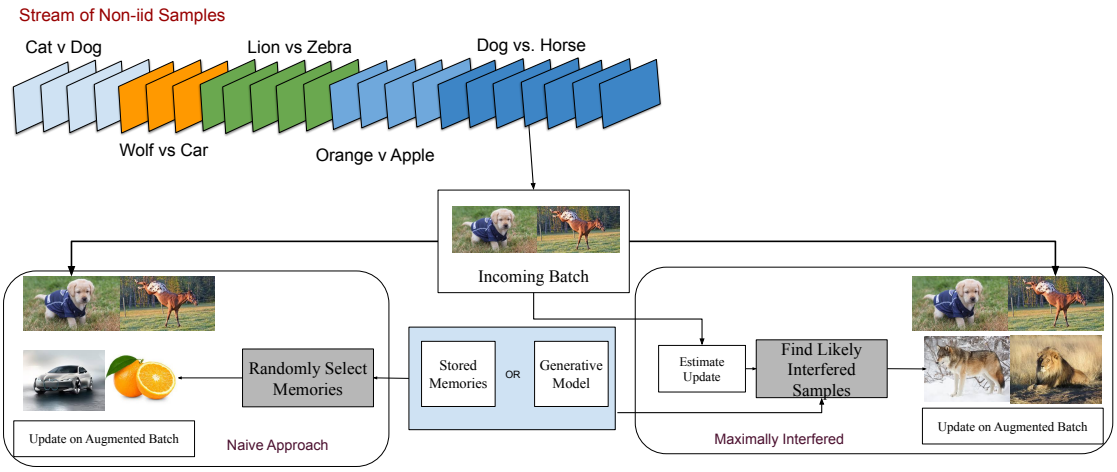

Figure 4.1 High-level illustration of a standard rehearsal method (left) such as generative replay or experience replay which selects samples randomly. This is contrasted with selecting samples based on interferences with the estimated update (right).

find samples that would be (maximally) interfered by the new incoming sample(s), had they been learned in isolation (Figure 4.1). This is motivated by the observation that the loss of some previous samples may be unaffected or even improved, thus retraining on them is wasteful. We formulate this first in the context of a small storage of past samples and subsequently using a latent variable generative model.

4.1.1 Maximally Interfered Sampling from a Replay Memory

We first instantiate our method in the context of experience replay (ER), a recent and successful rehearsal method [Chaudhry et al., 2019a], which stores a small subset of previous samples and uses them to augment the incoming data. In this approach the learner is allocated a memory $\mathcal{M}$ of finite size, which is updated by the use of reservoir sampling [Aljundi et al., 2019c, Chaudhry et al., 2019a] as the stream of samples arrives. Typically samples are drawn randomly from memory and concatenated with

the incoming batch.

Given a standard objective $\min_{\theta} \mathcal{L}(f_\theta(\boldsymbol{X}_t), \boldsymbol{Y}_t)$, when receiving sample(s) $\boldsymbol{X}_t$ we estimate the would-be parameters update from the incoming batch as

$$\theta^v = \theta - \alpha \nabla \mathcal{L}(f_\theta(\boldsymbol{X}_t), \boldsymbol{Y}_t),$$

with learning rate α. We can now search for the top-k values $x \in \mathcal{M}$ using the criterion

$$s_{MI\text{-}1}(x) = l(f_{\theta^v}(x), y) - l(f_\theta(x), y),$$

where l is the sample loss. We may also augment the memory to additionally store the best $l(f_\theta(x), y)$ observed so far for that sample, denoted $l(f_{\theta^*}(x), y)$. Thus instead we can evaluate $s_{MI\text{-}2}(x) = l(f_{\theta^v}(x), y) - \min\left(l(f_\theta(x), y), l(f_{\theta^*}(x), y)\right)$. We will consider both versions of this criterion in the sequel.

We denote the budget of samples to retrieve, $\mathcal{B}$. To encourage diversity we apply a simple strategy of performing an initial random sampling of the memory, selecting C samples where $C > \mathcal{B}$ before applying the search criterion. This also reduces the compute cost of the search. The ER algorithm with MIR is shown in Algorithm 2. We note that for the case of $s_{MI\text{-}2}$ the loss of the C selected samples at line 14 is tracked and stored as well.

4.1.2 Maximally Interfered Sampling from a Generative Model

We now consider the case of replay from a generative model. Assume a function f parameterized by θ (e.g. a classifier) and an encoder q_ϕ and decoder g_γ model parameterized by ϕ and γ, respectively. We can compute the would-be parameter update θ^v as in the previous section. We want to find in the given feature space data points that maximize the difference between their loss before and after the estimated parameters

Algorithm 2 Experience MIR (ER-MIR)

Input: Learning rate α, Subset size C; Budget $\mathcal{B}$

Initialize: Memory $\mathcal{M}$; θ **for** $t \in 1..T$ **do**

  for $B_n \sim D_t$ **do**

    $\theta^v \leftarrow \texttt{SGD}\,(B_n, \alpha)$ `// Virtual Update`

    $B_{\mathcal{C}} \sim \mathcal{M}$ `// Select C samples`

    $S \leftarrow sort(s_{MI}(B_{\mathcal{C}}))$ `// Select based on score`

    $B_{\mathcal{M}_{\mathcal{C}}} \leftarrow \{S_i\}_{i=1}^{\mathcal{B}}$ `// Aggregate interefered data`

    $\theta \leftarrow SGD(B_n \cup B_{\mathcal{M}_{\mathcal{C}}}, \alpha)$ `// Gradient Descent Step`

    $\mathcal{M} \leftarrow UpdateMemory(B_n)$ `// Add samples to memory`

update:

$$\begin{aligned} \max_{\boldsymbol{Z}} \quad & \mathcal{L}(f_{\theta^v}(g_\gamma(\boldsymbol{Z})), \boldsymbol{Y}^*) - \mathcal{L}(f_{\theta'}(g_\gamma(\boldsymbol{Z})), \boldsymbol{Y}^*) \\ \text{s.t.} \quad & ||z_i - z_j||_2^2 > \epsilon \quad \forall z_i, z_j \in \boldsymbol{Z} \text{ with } z_i \neq z_j \end{aligned} \tag{4.1}$$

with $Z \in \mathbb{R}^{\mathcal{B} \times \mathcal{K}}$, $\mathcal{K}$ the feature space dimension, and ϵ a threshold to encourage the diversity of the retrieved points. Here $\theta^{'}$ can correspond to the current model parameters or a historical model as in Shin et al. [2017]. Furthermore, y^* denotes the *true* label i.e. the one given to the generated sample by the real data distribution. We will explain how to approximate this value shortly. We convert the constraint into a regularizer and optimize the Equation 4.1 with stochastic gradient descent denoting the strength of the diversity term as λ. From these points we reconstruct the full corresponding input samples $\boldsymbol{X}^{'} = g_\gamma(Z)$ and use them to estimate the new parameters update $\min_\theta \mathcal{L}(f_\theta(\boldsymbol{X}_t \cup \boldsymbol{X}^{'}))$.

Algorithm 3 Generative-MIR (GEN-MIR)

Input: Learning rate α

Initialize: Memory $\mathcal{M}$; θ, ϕ,γ

for $t \in 1..T$ **do**

  $\theta', \phi', \gamma' \leftarrow \theta, \phi, \gamma$

  for $B_n \sim D_t$ **do**

    $\theta^v \leftarrow \texttt{SGD}\,(B_n, \alpha)$ `// Virtual Update`

    $B_C \leftarrow$ Retrieve samples as per `Eq (4.2)`

    $B_G \leftarrow$ Retrieve samples as per `Eq (4.3)`

    $\theta \leftarrow SGD(B_n \cup B_C, \alpha)$ `// Update Classifier`

    $\phi, \gamma \leftarrow SGD(B_n \cup B_G, \alpha)$ `// Update Generative Model`

Using the encoder encourages a better representation of the input samples where similar samples lie close. Our intuition is that the most interfered samples share features with new one(s) but have different labels. For example, in handwritten digit recognition, the digit 9 might be written similarly to some examples from digits {4,7}, hence learning 9 alone may result in confusing similar 4(s) and 7(s) with 9 (Fig. 4.2). The retrieval is initialized with $\boldsymbol{Z} \sim q_\phi(\boldsymbol{X}_t)$ and limited to a few gradient updates, limiting its footprint.

Figure 4.2 Most interfered retrieval from VAE on MNIST. Top row shows incoming data from a final task (8 v 9). The next rows show the samples causing most interference for the classifier (Eq. 4.1)

To estimate the loss in Eq. 4.1 we also need an estimate of y^* i.e. the label when using a generator. A straightforward approach for is based on the generative replay ideas [Shin et al., 2017] of storing the predictions of a prior model. We thus suggest to use the

predicted labels given by $f_{\theta'}$ as pseudo labels to estimate y^*. Denoting $y_{pre} = f_{\theta'}(g_\gamma(z))$ and $\hat{y} = f_{\theta^v}(g_\gamma(z))$ we compute the KL divergence, $D_{KL}(y_{pre} \parallel \hat{y})$, as a proxy for the interference.

Generative models such as VAEs Kingma and Welling [2014] are known to generate blurry images and images with mix of categories. To avoid such a source of noise in the optimization, we minimize an entropy penalty to encourage generating points for which the previous model is confident. The final objective of the generator based retrieval is

$$\begin{aligned} &\max_{\boldsymbol{Z}} \sum_{z \in \boldsymbol{Z}} [D_{KL}(y_{pre} \parallel \hat{y}) - \alpha H(y_{pre})] \\ &s.t. \quad ||z_i - z_j||_2^2 > \epsilon \; \forall z_i, z_j \in \boldsymbol{Z} \text{ with } z_i \neq z_j, \end{aligned} \tag{4.2}$$

with the entropy H and a hyperparameter α to weight the contribution of each term.

So far we have assumed having a perfect encoder/decoder that we use to retrieve the interfered samples from the previous history for the function being learned. Since we assume an online continual learning setting, we need to address learning the encoder/decoder continually as well. We could use a variational autoencoder (VAE) with $p_\gamma(X \mid z) = \mathcal{N}(X \mid g_\gamma(z), \sigma^2 I)$ with mean $g_\gamma(z)$ and covariance $\sigma^2 I$.

As for the classifier we can also update the VAE based on incoming samples and the replayed samples. In Eq. 4.1 we only retrieve samples that are going to be interfered given the classifier update, assuming a good feature representation. We can also use the same strategy to mitigate catastrophic forgetting in the generator by retrieving the most interfered samples given an estimated update of both parameters (ϕ, γ). In this case, the intereference is with respect to the VAE's loss, the evidence lower bound (ELBO). Let us denote γ^v, ϕ^v the virtual updates for the encoder and decoder given

the incoming batch. We consider the following criterion for retrieving samples for the generator:

$$\begin{aligned} \max_{\boldsymbol{Z}_{\text{gen}}} & \underset{z\sim q_{\phi^v}}{E}[-log(p_{\gamma^v}(g_{\gamma^v}(\boldsymbol{Z}_{\text{gen}})|z))] - \underset{z\sim q_{\phi'}}{E}[-log(p_{\gamma'}(g_{\gamma'}(\boldsymbol{Z}_{\text{gen}})|z))] \\ & + D_{KL}(q_{\phi^v}(z|g_{\gamma^v}(\boldsymbol{Z}_{\text{gen}}))||p(z)) - D_{KL}(q_{\phi'}(z|g_{\gamma'}(\boldsymbol{Z}_{\text{gen}}))||p(z)) \\ s.t. \quad & ||z_i - z_j||_2^2 > \epsilon \; \forall z_i, z_j \in \boldsymbol{Z}_{\text{gen}} \;\; \text{s.t.} \;\; z_i \neq z_j \end{aligned} \tag{4.3}$$

Here (ϕ', γ') can be the current VAE or stored from the end of the previous task. Similar to $\boldsymbol{Z}$, $\boldsymbol{Z}_{\text{gen}}$ is initialized with $Z_{\text{gen}} \sim q_\phi(X_t)$ and limited to few gradient updates.

4.1.3 A Hybrid Approach

Training generative models in the continual learning setting on more challenging datasets like CIFAR-10 remains an open research problem Lesort et al. [2019a]. Storing samples for replay is also problematic as it is constrained by storage costs and very-large memories can become difficult to search. To leverage the benefits of both worlds while avoiding training the complication of noisy generation, similar to Riemer et al. [2019] we use a hybrid approach where an autoencoder is first trained offline to store and compress incoming memories. Differently, in our approach, we perform MIR search in the latent space of the autoencoder using Eq. 4.1. We then select nearest neighbors from stored compressed memories to ensure realistic samples. Our strategy has several benefits: by storing lightweight representations, the buffer can store more data for the same fixed amount of memory. Moreover, the feature space in which encoded samples lie is fully differentiable. This enables the use of gradient methods to search for most interfered samples. We note that this is not the case for the discrete autoencoder proposed in Riemer et al. [2019]. Finally, the autoencoder with its simpler objective is easier to train in the online setting than a variational autoencoder.

4.2 Experiments

We now evaluate the proposed method under the generative and experience replay settings. We use three standard datasets and the shared classifier setting described below.

- **MNIST Split** splits MNIST data to create 5 different tasks with non-overlapping classes. We consider the setting with 1000 samples per task as in Aljundi et al. [2019b], Lopez-Paz and Ranzato [2017].
- **Permuted MNIST** permutes MNIST to create 10 different tasks. We consider the setting with 1000 samples per task as in Aljundi et al. [2019b], Lopez-Paz and Ranzato [2017].
- **CIFAR-10 Split** splits CIFAR-10 dataset into 5 disjoint tasks as in Aljundi et al. [2019c]. However, we use a more challenging setting, with all 9,750 samples per task and 250 retained for validation.
- **MiniImagenet Split** splits MiniImagenet Vinyals et al. [2016] dataset into 20 disjoint tasks as in Chaudhry et al. [2019a] with 5 classes each.

In our evaluations we focus the comparisons of MIR to random sampling in the experience replay (ER) [Aljundi et al., 2019c, Chaudhry et al., 2019a] and generative replay [Shin et al., 2017, Lavda et al., 2018] approaches which our method directly modifies. We also consider the following reference baselines:

- **fine-tuning** trains continuously upon arrival of new tasks without any forgetting avoidance strategy.
- **iid online** (upper-bound) considers training the model with a single-pass through the data on the same set of samples, but sampled iid.

- **iid offline** (upper-bound) evaluates the model using multiple passes through the data, sampled iid. We use 5 epochs in all the experiments for this baseline.
- **GEM** Lopez-Paz and Ranzato [2017] is another method that relies on storing samples and has been shown to be a strong baseline in the online setting. It gives similar results to the recent A-GEM Chaudhry et al..

We do not consider prior-based baselines such as Kirkpatrick et al. [2017] as they have been shown to work poorly in the online setting as compared to GEM and ER [Chaudhry et al., 2019a, Lopez-Paz and Ranzato, 2017]. For evaluation we primarily use the accuracy as well as forgetting [Chaudhry et al., 2019a].

Shared Classifier A common setting for continual learning applies a separate classifier for each task. This does not cover some of the potentially more interesting continual learning scenarios where task metadata is not available at inference time and the model must decide which classes correspond to the input from all possible outputs. As in Aljundi et al. [2019c] we adopt a shared-classifier setup for our experiments where the model can potentially predict all classes from all tasks. This sort of setup is more challenging, yet can apply to many realistic scenarios.

Multiple Updates for Incoming Samples In the one-pass through the data continual learning setup, previous work has been largely restricted to performing only a single gradient update on incoming samples. However, as in Aljundi et al. [2019c] we argue this is not a necessary constraint as the prescribed scenario should permit maximally using the current sample. In particular for replay methods, performing additional gradient updates with additional replay samples can improve performance. In the sequel we will refer to this as performing more iterations.

Comparisons to Reported Results Comparing reported results in Continual Learning requires great diligence because of the plethora of experimental settings. We remind the reader that our setting, i.e. shared-classifier, online and (in some cases) lower amount of training data, is more challenging than many of the other reported continual learning settings. Therefore, the reader must be vigilant in comparing results across equivalent settings.

4.2.1 Experience Replay

Here we evaluate experience replay with MIR comparing it to standard experience replay [Chaudhry et al., 2019a, Aljundi et al., 2019c] on a number of shared classifier settings. In all cases we use a single update for each incoming batch, multiple iterations/updates are evaluated in a final ablation study. We restrict ourselves to the use of reservoir sampling for deciding which samples to store. We first evaluate using the MNIST Split and Permuted MNIST (Table 4.1). We use the same learning rate, 0.05, used in Aljundi et al. [2019c]. The number of samples from the replay buffer is always fixed to the same amount as the incoming samples, 10, as in Chaudhry et al. [2019a]. For MIR we select by validation $C = 50$ and the $s_{MI\text{-}2}$ criterion for both MNIST datasets. ER-MIR performs well and improves over (standard) ER in both accuracy and forgetting. We also show the accuracy on seen tasks after each task sequence is completed in Figure 4.7.

We now consider the more complex setting of CIFAR-10 and use a larger number of samples than in prior work Aljundi et al. [2019c]. We study the performance for different memory sizes (Table 4.2). For MIR we select by validation at $M = 50$, $C = 50$ and the $s_{MI\text{-}1}$ criterion. We observe that the performance gap increases when more memories are used. We find that the GEM method does not perform well in this setting. We also consider another baseline iCarl Rebuffi et al. [2017]. Here we boost the iCarl method permitting it to perform 5 iterations for each incoming sample to maximize its

performance. Even in this setting it is only able to match the experience replay baseline and is outperformed by ER-MIR for larger buffers.

method	Split MNIST		Permuted MNIST	
	Accuracy (↑)	Forgetting (↓)	Accuracy (↑)	Forgetting (↓)
iid online	86.8 ± 1.1	N/A	73.8 ± 1.2	N/A
iid offline	92.3 ± 0.5	N/A	86.6 ± 0.5	N/A
fine-tuning	19.0 ± 0.2	97.8 ± 0.2	64.6 ± 1.7	15.2 ± 1.9
GEN	79.3 ± 0.6	19.5 ± 0.8	79.7 ± 0.1	5.8 ± 0.2
GEN-MIR	$\mathbf{82.1 \pm 0.3}$	$\mathbf{17.0 \pm 0.4}$	$\mathbf{80.4 \pm 0.2}$	$\mathbf{4.8 \pm 0.2}$
GEM	86.3 ± 1.4	11.2 ± 1.2	78.8 ± 0.4	$\mathbf{3.1 \pm 0.5}$
ER	82.1 ± 1.5	15.0 ± 2.1	78.9 ± 0.6	3.8 ± 0.6
ER-MIR	$\mathbf{87.6 \pm 0.7}$	$\mathbf{7.0 \pm 0.9}$	$\mathbf{80.1 \pm 0.4}$	3.9 ± 0.3

Table 4.1 Results for MNIST SPLIT (left) and Permuted MNIST (right). We report the Average Accuracy (higher is better) and Average Forgetting (lower is better) after the final task, along with the standard error. We split results into privileged baselines, methods that don't use a memory storage, and those that store memories. For the ER methods, 50 memories per class are allowed. Each approach is run 20 times.

Increased iterations We evaluate the use of additional iterations on incoming batches by comparing the 1 iteration results above to running 5 iterations. Results are shown in Table 4.3. We use ER an and at each iteration we either re-sample randomly or using the MIR criterion. We observe that increasing the number of updates for an incoming sample can improve results on both methods.

Longer Tasks Sequence Next, we want to test how our strategy performs on longer sequences of tasks. For this we consider the 20 tasks sequence of MiniImagenet Split. Note that this dataset is very challenging in our setting given the shared classifier and the online training. A naive experience replay with 100 memories per class obtains only 17% accuracy at the end of the task sequence. To overcome this difficulty, we allow more iterations per incoming batch. Table 4.4 compares ER and ER-MIR accuracy and

	Accuracy (↑)			Forgetting ↓		
	$M = 20$	$M = 50$	$M = 100$	$M = 20$	$M = 50$	$M = 100$
iid online	60.8±1.0	60.8±1.0	60.8±1.0	N/A	N/A	N/A
iid offline	79.2±0.4	79.2±0.4	79.2±0.4	N/A	N/A	N/A
GEM	16.8±1.1	17.1±1.0	17.5±1.6	73.5±1.7	70.7±4.5	71.7±1.3
iCarl (5 iter)	28.6±1.2	33.7±1.6	32.4±2.1	**49**±2.4	40.6±1.1	40±1.8
fine-tuning	18.4±0.3	18.4±0.3	18.4±0.3	85.4±0.7	85.4±0.7	85.4±0.7
ER	27.5±1.2	33.1±1.7	41.3±1.9	50.5±2.4	35.4±2.0	23.3±2.9
ER-MIR	**29.8**±1.1	**40.0**±1.1	**47.6**±1.1	50.2±2.0	**30.2**±2.3	**17.4**±2.1

Table 4.2 CIFAR-10 results. Memories per class M, we report (a) Accuracy, (b) Forgetting (lower is better) and their corresponding standard error. For larger sizes of memory ER-MIR has better accuracy and improved forgetting metric. Each approach is run 15 times.

	Number of iterations	
	1	5
iid online	60.8 ± 1.0	62.0 ± 0.9
ER	41.3 ± 1.9	42.4 ± 1.1
ER-MIR	47.6 ± 1.1	49.3 ± 0.1

Table 4.3 CIFAR-10 accuracy (↑) results for increased iterations and 100 memories per class. Each approach is run 15 times, standard erorr is reported.

	Accuracy ↑	Forgetting ↓
ER	24.7 ± 0.7	23.5 ± 1.0
ER-MIR	25.2±0.6	18.0±0.8

Table 4.4 MinImagenet results. 100 memories per class and using 3 updates per incoming batch, accuracy is slightly better and forgetting is greatly improved. Each approach is run 15 times, standard error is reported.

forgetting at the end of the sequence. It can be seen how our strategy continues to outperform, in particular we achieve over 5% decrease in forgetting.

4.2.2 Generative Replay

We now study the effect of our proposed retrieval mechanism in the generative replay setting (Alg. 3). Recall that online continual generative modeling is particularly challenging and to the best of our knowledge has never been attempted. This is further

exacerbated by the low data regime we consider.

Results for the MNIST datasets are presented in Table 4.1. To maximally use the incoming samples, we (hyper)-parameter searched the amount of additional iterations for both GEN and GEN-MIR. In that way, both methodologies are allowed their optimal performance. More hyperarameter details are provided in the Appendix. On MNIST Split, MIR outperforms the baseline by 2.8% and 2.5% on accuracy and forgetting respectively. Methods using stored memory show improved performance, but with greater storage overhead. We provide further insight into theses results with a generation comparison (Figure 4.3). Complications arising from online generative modeling combined with the low data regime cause blurry and/or fading digits (Figure 4.3a) in the VAE baseline (GEN). In line with the reported results, the most interfered retrievals seem qualitatively superior (see Figure 4.3b where the GEN-MIR generation retrievals is demonstrated). We note that the quality of the samples causing most interference on the VAE seems higher than those on the classifier.

(a) Generation with the best VAE baseline. Complications arising from both properties leave the VAE generating blurry and/or fading digits.

(b) Most interfered samples while learning the last task (8 vs 9). Top row is the incoming batch. Rows 2 and 3 show the most interfered samples for the classifier, Row 4 and 5 for the VAE. We observe retrieved samples look similar but belong to different category.

Figure 4.3 Online and low data regime MNIST Split generation. Qualitatively speaking, most interfered samples are superior to baseline's.

For the Permuted MNIST dataset, GEN-MIR not only outperforms the baselines, but it achieves the best performance over all models. This result is quite interesting, as

generative replay methods can't store past data and require much more tuning.

The results discussed thus far concern classification. Nevertheless, GEN-MIR alleviates catastrophic forgetting in the generator as well. Table 4.5 shows results for the online continual generative modeling. The loss of the generator is significantly lower on both datasets when it rehearses on maximally interfered samples versus on random samples. This result suggests that our method is not only viable in supervised learning, but in generative modeling as well.

	MNIST Split	Permuted MNIST
GEN	107.2 ± 0.2	196.7 ± 0.7
GEN-MIR	$\mathbf{102.5 \pm 0.2}$	$\mathbf{193.7 \pm 1.0}$

Table 4.5 Generator's loss (↓), i.e. negative ELBO, on the MNIST datasets. Our methodology outperforms the baseline in online continual generative modeling as well. We report standard error.

Our last generative replay experiment is an ablation study. The results are presented in Table 4.6. All facets of our proposed methodology seem to help in achieving the best possible results. It seems however that the minimization of the label entropy, i.e. $H(y_{pre})$, which ensures that the previous classifier is confident about the retrieved sample's class, is most important and is essential to outperform the baseline.

As noted in Lesort et al. [2019a], training generative models in the continual learning setting on more challenging datasets remains an open research problem. Lesort et al. [2019a] found that generative replay is not yet a viable strategy for CIFAR-10 given the current state of the generative modeling[1]. We too arrived at the same conclusion, which led us to design the hybrid approach presented next.

4.2.3 Hybrid Approach

[1] this paper was published in 2019. See section 4.5 for relation to the more recent literature.

	Accuracy
GEN-MIR	83.0
ablate MIR on generator	82.7
ablate MIR on classifier	81.7
ablate $D_{KL}(y_{pre} \parallel \hat{y})$	80.7
ablate $H(y_{pre})$	78.3
ablate diversity constraint	80.7
GEN	80.0

Table 4.6 Ablation study of GEN-MIR on the MNIST Split dataset. The $H(y_{pre})$ term in the MIR loss function seems to play an important role in the success of our method.

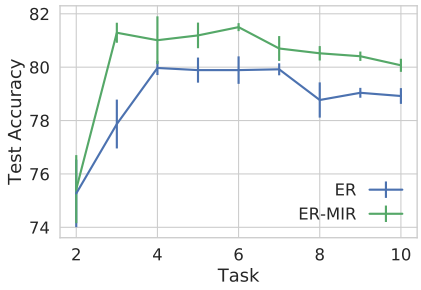

Table 4.7 Permuted MNIST test accuracy on tasks seen so far for rehearsal methods. Error bars denote standard error.

In this section, we evaluate the hybrid approach proposed in Sec 4.1.3 on the CIFAR-10 dataset. We use an autoencoder to compress the data stream and simplify MIR search.

We first identify an important failure mode arising from the use of reconstructions which may also apply to generative replay. During training, the classifier sees real images, from the current task, from the data stream, along with reconstructions from the buffer, which belong to old tasks. In the shared classifier setting, this discrepancy can be leveraged by the classifier as a discriminative feature. The classifier will tend to classify all real samples as belonging to the classes of the last task, yielding low test accuracy. To address this problem, we first autoencode the incoming data with the generator before passing it to the classifier. This way, the classifier cannot leverage the

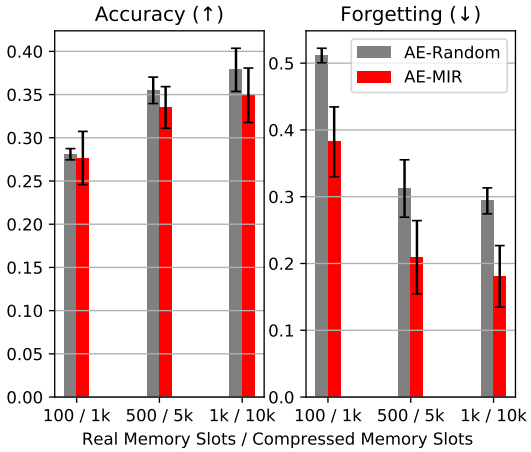

Figure 4.4 Results for the Hybrid Approach. Error bars denote standard error.

distribution shift. We found that this simple correction led to a significant performance increase. We perform an ablation experiment to validate this claim, which can be found in the Appendix , along with further details about the training procedure.

In practice, we store a latent representation of size $4 \times 4 \times 20 = 320$, giving us a compression factor of $\frac{32 \times 32 \times 3}{320} = 9.6$ (putting aside the size of the autoencoder, which is less than 2% of total parameters for large buffer size). We therefore look at buffer size which are 10 times as big i.e. which can contain 1k, 5k, 10k compressed images, while holding memory equivalent to storing 100 / 5000 / 1k real images. Results are shown in Figure 4.4. We first note that as the number of compressed samples increases we continue to see performance improvement, suggesting the increased storage capacity gained from the autoencoder can be leveraged. We next observe that even though AE-MIR obtains almost the same average accuracies as AE-Random, it achieved a big decrease in the forgetting metric, indicating a better trade-offs in the performance of the learned tasks. Finally we note a gap still exists between the performance of reconstructions from incrementally learned AE or VAE models and real images, further work is needed to close it.

4.3 Discussion

In this chapter, we have proposed and studied a criterion for retrieving relevant memories in an online continual learning setting. We have shown in a number of settings that retrieving interfered samples reduces forgetting and significantly improves on random sampling and standard baselines. Our results and analysis also shed light on the feasibility and challenges of using generative modeling in the online continual learning setting. We have also shown a first result in leveraging encoded memories for more compact memory and more efficient retrieval.

A first drawback of the analysis presented in this chapter is the lack of proper mon-

itoring of computation during training. Indeed, the MIR selection criterion requires additional forward-backward passes through the model, which amounts to a $2\times$ increase of FLOPs during training, as we will see in chapter 6. Nevertheless, we have shown that increasing the number of training iterations of standard replay, effectively giving ER a compute budget superior to MIR, does not enable it to surpass our proposed method.

Moreover, given the limited scope of the evaluation setup, which primarily concentrated on online, image-based class-incremental learning scenarios with small memory requirements, it remains uncertain how MIR would perform in more diverse and realistic settings (such as starting from a pretrained model). Consequently, readers are advised to exercise caution when extrapolating these results to broader contexts.

4.4 Follow-up findings in the community

Subsequent to the publishing of this paper, Shim et al. [2021] conducted an in-depth survey of several Continual Learning methods, including MIR, in both class-incremental and domain-incremental settings. In the latter, the marginal distribution $p(y)$ is fixed, i.e. the same classes are observed across tasks, however $p(x|y)$ changes for each task. The authors also evaluated on additional datasets, and found that "MIR performs the best in larger-scale datasets, [...] is overall a strong and versatile online continual learning method across a wide variety of settings", and that "[...] MIR produces performance levels that bring online continual learning much closer to its ultimate goal of matching offline training." These findings suggest that in larger-scale settings, where agents can potentially afford to store all previously seen data, MIR could be an effective strategy to carefully choose rehearsal samples. On the other hand, it was shown in Masana et al. [2022] that applying MIR to offline class-incremental learning scenarios does not yield any gains w.r.t to standard replay, suggesting that MIR's use may be limited to online

settings.

Finally, several works have also investigated the question of what samples should be used for replay. Shim et al. [2021] proposed an Adversarial Shapley Value scoring method which can be used to select which samples to store in the buffer, as well as which ones to rehearse on. They showed that this approach leads to more diverse sample selection than MIR. Nisar et al. [2023] extended the MIR approach to retrieved samples whose gradients interfere the most with the gradient of the loss computed on the incoming data.

Chapter 5

Online Continual Compression via Adaptive Quantization Modules

This chapter focuses on the following familiar setting: new training data arrives continuously for a learning algorithm to exploit, however this data might not be iid, and furthermore there is insufficient storage capacity to preserve all the data uncompressed. We may want to train classifiers, reinforcement learning policies, or other models continuously from this data as it's being collected, or use samples randomly drawn from it at a later point for a downstream task. For example, an autonomous vehicle (with bounded memory) collects large amounts of high-dimensional training data (video, 3D lidar) in a non-stationary environment (e.g. changing weather conditions), and over time applies an ML algorithm to improve its behavior using this data. This data might be transferred at a later point for use in downstream learning. Current learned compression algorithms, e.g. Torfason et al. [2018], are not well designed to deal with this case, as their convergence speed is too slow to be usable in an online setting.

In the previous chapter, we have shown that for Continual Learning approaches based on storing memories for later use have emerged as some of the most effective in online settings [Lopez-Paz and Ranzato, 2017, Aljundi et al., 2019c, Chaudhry

et al., 2019a, Aljundi et al., 2019a]. We saw that these memories can be stored as is, or via a generative model [Shin et al., 2017]. Indeed, many continual learning applications would be greatly improved with replay approaches if one could afford to store all samples. These approaches are however inherently limited by the amount of data that can be stored. Learning a generative model to compress the previous data stream thus seems like an appealing idea. However, as demonstrated in the previous chapter, training generative models in the online and non-stationary setting continues to be challenging, and can greatly increase the complexity of the continual learning task. Furthermore, such models are susceptible to catastrophic forgetting. This led us to propose a hybrid solution, where a compressed representation of the data is learned, which can be more stable than learning generative models. While the approach showed some promise, several limitations need to be addressed for the hybrid method to be a viable alternative to standard replay. Specifically, (i) offline pretraining of the compressor was required, (ii) there was a clear distribution shift between raw and reconstructed images, and (iii) overall this approach underperformed standard replay in online class-incremental settings for the same memory budget.

In the following chapter, we propose a new approach for online continual compression, which addresses all three challenges above. Moreover, we show that when dealing with long sequences, learned compression in the online and non-stationary setting itself introduces new challenge, illustrated in Fig 5.2. Firstly the learned compression module must be able to decode representations encoded by earlier versions of itself, introducing a problem we refer to as *representation drift*. Secondly, the learned compressor is itself susceptible to catastrophic forgetting. Finally, the learned compression needs to be adaptive to maintain a prescribed level of reconstruction quality even if it has not fully adapted to the current state of the distribution.

In this work we demonstrate that the VQ-VAE framework [Van Den Oord et al., 2017, Razavi et al., 2019], originally introduced in the context of generative model-

ing and density estimation, can be used online to effectively address representation drift while achieving high compression. Furthermore, when augmented with an internal replay mechanism it can overcome forgetting. Finally we propose to use multiple gradient-isolated compression levels to allow the compressor to adaptively store samples at different compression scales, based on the amount of data, storage capacity, and effectiveness of the model in compressing samples.

Thus, in this chapter, I discuss our contributions in Caccia et al. [2020a], which are as follows: (i) we introduce and highlight the online learned continual compression (OCC) problem and its challenges. (ii) We show how representation drift, one of the key challenges, can be tackled by effective use of codebooks in the VQ-VAE framework. (iii) We propose an architecture using multiple VQ-VAEs, adaptive compression scheme, stream sampling scheme, and self-replay mechanism that work together to effectively tackle the OCC problem. (iv) We demonstrate this can yield state-of-the-art performance in standard online continual image classification benchmarks and demonstrate the applications of our OCC solution in a variety of other contexts.

5.1 Technical Background

5.1.1 Vector Quantized Variational Autoencoder

Variational Autoencoders (VAE) [Kingma and Welling, 2014] can be seen as a regularized version of the standard autoencoder, making VAEs valid generative models. They consist of two parts: the encoder network parameterizes the posterior distribution $q(z|x)$ and the decoder network $p(x|z)$ aims to reconstruct the original input x from the inferred latent variables z. In a standard VAE, the prior and posterior are modeled as an isotropic Gaussian distribution. On the other hand, Vector Quantized Autoencoders (VQ-VAE) use a discrete latent representation instead [Van Den Oord et al., 2017]. The VQ-VAE is a discrete auto-encoder which relies on a vector quanti-

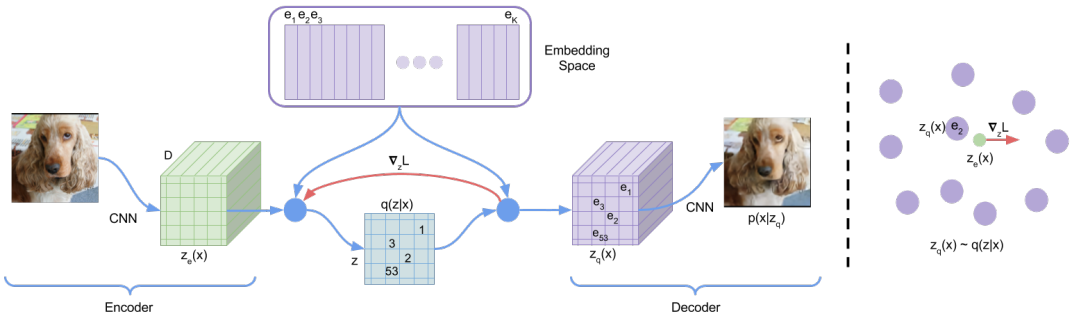

Figure 5.1 *left :* VQ-VAE workflow, taken from Van Den Oord et al. [2017]. The red arrow denotes the use of the straight-through estimator. *right :* Visualisation of the embedding space, with the green dot denoting the encoder output and e_2 denoting its quantized representation.

zation step to obtain discrete latent representations. An embedding table, $E \in \mathbb{R}^{K \times D}$ consisting of K vectors of size D, is used to quantize encoder outputs. Given an input (e.g. an RGB image), the encoder first encodes it as a $H_h \times W_h \times D$ tensor, where H_h and W_h denote the height and width of the latent representation. Then, every D dimensional vector is quantized using a nearest-neighbor lookup on the embedding table. Specifically,

$$z_{ij} = \arg\min_{e \in E} ||\texttt{enc}(x)_{ij} - e||_2,$$

where i, j refers to a spatial location. The output of the quantization step is then fed through the decoder. The gradient of this non-differentiable step is approximated using the straight-through estimator. An important property to notice is that to reconstruct the input, only the $H_h \times W_h$ indices are required, thus yielding high compression [Van Den Oord et al., 2017].

Critically, the embedding tables are updated independently from the encoder and decoder, namely by minimizing $\min_e ||sg[\texttt{enc}(x)_{ij}] - e||$, where sg is the stop gradient operator.

5.1.2 LiDAR data in autonomous driving

LiDAR (Light Detection and Ranging) is a remote sensing technology that uses laser light to measure distances and generate detailed, accurate maps of the environment. In the context of autonomous driving, LiDAR sensors are mounted on top of vehicles to provide high-resolution 3D point clouds, which can be used for object detection, localization, and mapping. LiDAR data is particularly valuable for autonomous vehicles, as it can provide accurate depth information, allowing the vehicles to safely navigate their surroundings. We follow our previous work in Caccia et al. [2019], and encode lidar scans in a 2D grid, which can then be processed using standard convolutional neural networks.

5.2 Methodology

In this section we outline our approach to the online continual compression problem. First we review the VQ-VAE and highlight the properties making it effective for representational drift. Then we describe our adaptive architecture, storage, and sampling scheme.

5.2.1 Problem Setting: Online Continual Compression

We consider the problem setting where a stream of samples $x \sim D_t$ arrives from different distributions D_t changing over time $t = 1 \dots T$. We have a fixed storage capacity of C bytes where we would like to store the most representative information from all data distributions $D_1, ...D_T$. There is notably a trade-off in quality of information versus the amount of samples stored. We propose to use a learned compression model, and most crucially, this model must also be stored within the C bytes, to encode and decode the data samples. Another critical requirement is that at anytime t the content of the storage (data and/or compression model) be usable for downstream applications. An

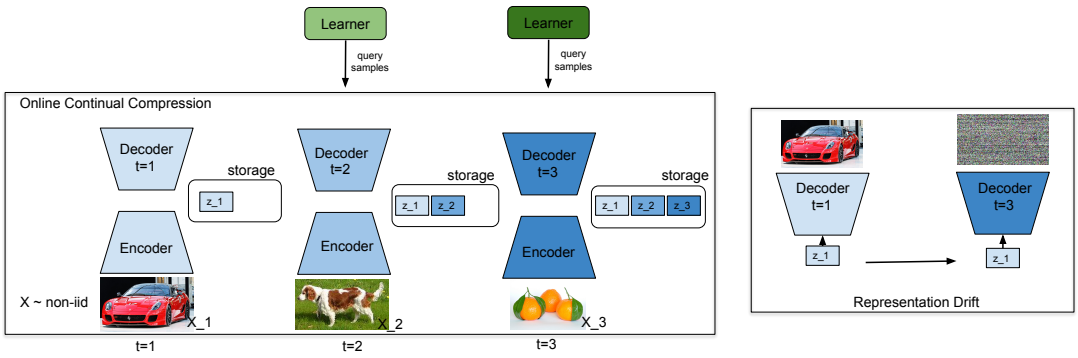

Figure 5.2 Illustration of the challenges in the Online Continual Compression problem. A model must be able to decode representations encoded by previous versions of the autoencoder, permitting anytime access to data for the learner. This must be accomplished while dealing with a time-varying data distribution and fixed memory constraints

important challenge, illustrated in Figure 5.2, is that the learned compression module will change over time, while we still need to be able to decode the memories in storage.

5.2.2 Vector Quantized VAE for Online Compression

Observe in the case of online compression, if the embedding table of the VQ VAE is fixed, then a change in the encoder parameters and therefore a change in the encoder output for a given input will not change the final quantized representation z, unless it is sufficiently large, thus we can observe that if the embeddings change slowly or are fixed we can greatly improve our control of the representa-

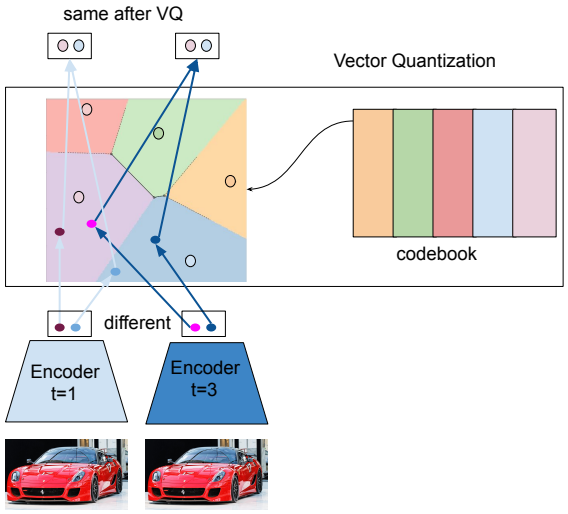

Figure 5.3 Illustration of reduced representation drift from Vector Quantization

Algorithm 4 AQM LEARNING WITH SELF-REPLAY

Input: Learning rate α, EXTERNALLEARNER
Initialize: AQM Memory $\mathcal{M}$; AQM Parameters θ_{aqm}
for $t \in 1..T$ **do**
  for $B_{inc} \sim D_t$ **do**
    for $n \in 1..N$ **do**
      $B \leftarrow B_{inc}$ `// Fetch data from current task`
      if $t > 1$ **then**
        $B_{re} \sim \text{SAMPLE}(M, \theta_{aqm})$ `// Fetch data from buffer`
        $B \leftarrow (B_{inc}, B_{re})$
      $\theta_{aqm} \leftarrow ADAM(\theta_{aqm}, B, \alpha)$ `// Update AQM`
      EXTERNALLEARNER(B) `// Send data to external learner`
      if $t > 1$ **then** UPDATEBUFFERREP(M, θ_{aqm})
      ADDTOMEMORY$(M, B_{inc}, \theta_{aqm})$ `// Save current indices`

tional drift. This effect is illustrated in Figure 5.3. On the other hand we do need to adapt the embedding table, since randomly selected embeddings would not cover well the space of encoder outputs.

5.2.3 Adaptive Quantization Modules

To address issues of how to optimize storage and sampling in the context of Online Continual Compression we introduce Adaptive Quantization Modules (AQM). We use AQM to collectively describe the architecture, adaptive multi-level storage mechanism, and data sampling method used. Together they allow effectively constraining individual sample quality and memory usage while keeping in storage a faithful representation of the overall distribution.

AQM uses an architecture consisting of a sequence of VQ-VAEs, each with a buffer. The AQM approach to online continual compression is overall summarized in Algorithm 4 (note ADDTOMEMORY is described in the appendix). Incoming data is added

to storage using an adaptive compression scheme described in Algorithm 5. Incoming data is also used along with randomly sampled data from storage (self-replay) to update the current AQM model. The randomly sampled data also updates its representation in storage as per Algorithm 6. As illustrated by the optional lines in blue, Algorithm 4 can run concurrently with a downstream learning task (e.g. online continual classification) which would use the same batch order. It can also be run independently as part of a data collection. In the sequel we give further details on all these elements

Architecture and Training

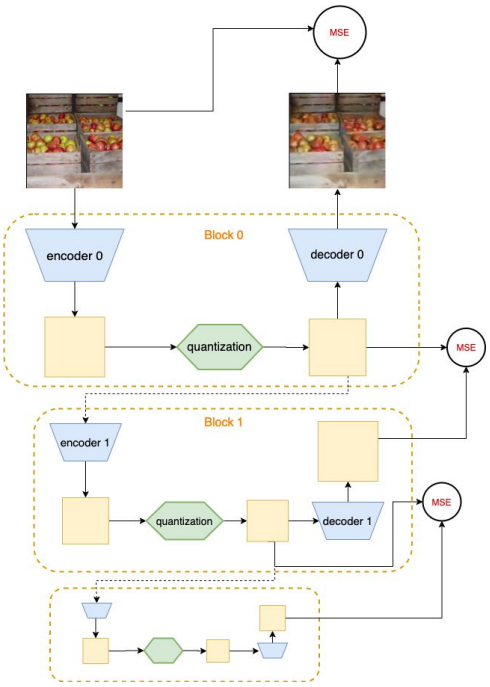

Figure 5.4 Architecture of Adaptive Quantization Modules. Each level uses its own loss and maintains its own replay buffer. Yello dotted lines indicate gradient isolation between modules

Each AQM module contains a VQ-VAE and a corresponding buffer of adaptive capacity. A diagram of the architecture is given in Figure 5.4. We will denote the output after quantization of each module i as z_q^i and the set of codebook indexes used to obtain z_q^i as a^i. Note that a^i are the discrete representations we actually store. Each subsequent module produces and stores an a^i requiring fewer bits to represent.

For RGB images, the compression rate at a given level is given by $\frac{H \times W \times 3 \times \log_2(256)}{N_c \times H_{hi} \times W_{hi} \times \lceil \log_2(K_i) \rceil}$. Here K_i is the number of embeddings in the codebooks, (H_{hi}, W_{hi}) the spatial dimension of the latent representation and N_{ci} the number of codebooks.

Algorithm 5 ADAPTIVECOMPRESS

Input: datapoint x, AQM with L modules, threshold d_{th}

$\{z_q^i, a^i\}_{i=1..L}$= ENCODE(x) `// Forward all modules, store encodings`

for $i \in L...1$ **do**

  $\hat{x} =$ DECODE(z_q^i) `// Decode from level i to output space`

  if MSE$(\hat{x}, x) < d_{th}$ **then** `// Check reconstruction error`

    return $a_i,\ i$

return $x, 0$ `// Otherwise, return original input`

VQVAE-2 [Razavi et al., 2019] also uses a multi-scale hierarchical organization, where unlike our AQM, the top level models global information such as shape, while the bottom level, conditioned on the top one, models local information. While this architecture is tailored for generative modeling, it is less attractive for compression, as both the bottom and top quantized representations must be stored for high quality reconstructions. Furthermore in AQM each module is learned in a greedy manner using the current estimate of $z_q^{(i-1)}$ without passing gradients between modules similar to Belilovsky et al. [2019], Nøkland and Eidnes [2019]. A subsequent module is not required to build representations which account for all levels of compression, thus minimizing interference across resolutions. This allows the modules to each converge as quickly as possible with minimal drift at their respective resolution, particularly important in the online continual learning case.

Multi-Level Storage

Our aim is to store the maximum number of samples in an allotted C bytes of storage, while assuring their quality, and our ability to reconstruct them. Samples are thus stored at different levels based on the compressors' current ability. The process is sum-

Algorithm 6 UpdateBufferRep

Input: Memory $\mathcal{M}$, AQM with L levels, data D, distortion threshold d_{th}

for $x \in D$ **do**

    hid_x, $block_{id}$ = AdaptiveCompress(x, AQM, d_{th})

    Delete($\mathcal{M}[x]$) // Delete Old Representation

    Add($\mathcal{M}$, hid_x) // Add new one

marized in Algorithm 5.

Such an approach is particularly helpful in the online non-stationary setting, allowing knowledge retention before the compressor network has learned well the current distribution. Note in Alg. 5 samples can be completely uncompressed until the first module is able to effectively encode them. This can be crucial in some cases, if the compressor has not yet converged, to avoid storing poorly compressed representations. Further taking into account that compression difficulty is not the same for all datapoints, this allows use of more capacity for harder data, and fewer for easier.

We also note, since we maintain stored samples at each module and the modules are decoupled, that such an approach allows to easily distribute training in an asynchronous manner as per Belilovsky et al. [2019].

Self-Replay and Stream Sampling

As shown in Alg. 4 our AQM is equipped with an internal experience replay mechanism [Mnih et al., 2013], which reconstructs a random sample from storage and uses it to perform an update to the AQM modules, while simultaneously freeing up overall memory if the sample can now be compressed at a later AQM module. This has the effect of both reducing forgetting and freeing memory. In practice we replay at the same rate as incoming samples arrive and thus replay will not increase asymptotic

complexity of the online learning. Finally, for efficiency the replay can be coupled to an external online learner querying for random samples from the overall memory.

Since we would like the AQM to work in cases of a fixed memory capacity it must also be equipped with a mechanism for selecting which samples from the stream to store and which to delete from memory. Reservoir Sampling (RS) is a simple yet powerful approach to this problem, used successfully in continual learning [Chaudhry et al., 2019a]. It adds a sample from the stream with prob. $p = \frac{\text{buffer capacity}}{\text{points seen so far}}$ while removing a random sample. However, RS is not directly compatible with AQM primarily because the amount of samples that can be stored varies over time. This is because samples at different levels have different memory usage and memory can be freed by replay. We thus propose an alternative scheme, which maximally fills available memory and selects non-uniformly samples for deletion. Specifically when a larger amount of samples are added at one point in the stream, they become more likely to be removed. The details of this stream sampling method are provided in the appendix.

Drift Control via Codebook Stabilization

As mentioned previously, a good online compressor must control its representational drift, which occurs when updates in the auto-encoder parameters creates a mismatch with the static representations in the buffer. Throughout this chapter we measure representational drift by comparing the following time varying quantity: $\text{DRIFT}_t(z) = \text{RECON ERR}(\texttt{Decode}(\theta_t; z), x)$ where θ_t the model parameters at time t and (z, x) is a stored compressed representation and its original uncompressed datapoint respectively. For all experiments on images, RECON ERR is simply the mean squared error.

As illustrated in Sec 5.2.2 a slow changing codebook can allow to control drifting representations. This can be in part accomplished by updating the codebook with an exponential moving average as described in [Van Den Oord et al., 2017, Appendix A], where it was used to reduce the variance of codebook updates. This alone is insuffi-

cient to fully control drift, thus once a given module yields satisfactory compressions on the data stream, we freeze the module's embedding matrix but leave encoder and decoder parameters free to change and adapt to new data. Moreover, we note that fixing the codebook for a given module does not affect the reconstruction performance of subsequent modules, as they only need access to the current module's decoder which can still freely change.

5.3 Experiments

We evaluate the efficacy of the proposed methods on a suite of canonical and new experiments. In Section 5.3.1 we present results on standard supervised continual learning benchmarks on CIFAR-10. In Section 5.3.2 we evaluate other downstream tasks such as standard iid training applied on the storage at the end of online continual compression. For this evaluation we consider larger images from Imagenet, as well as on lidar data. Finally we apply AQM on observations of an agent in an RL environment.

5.3.1 Online Continual Classification

Although CL has been studied in generative modeling [Ramapuram et al., 2017, Lesort et al., 2019a, Zhai et al., 2019, Lesort et al., 2019b] and reinforcement learning [Kirkpatrick et al., 2017, Fernando et al., 2017, Riemer et al., 2018], supervised learning is still the standard for evaluation of new methods. Thus, we focus on the online continual classification of images for which our approach can provide a complement to experience replay. In this setting, a new task consists of new image classes that the classifier must learn, while not forgetting the previous ones. The model is only allowed one pass through the data [Lopez-Paz and Ranzato, 2017, Chaudhry et al., Aljundi et al., 2019a, Chaudhry et al., 2019a]. The online compression here takes the role of replay buffer in replay based methods such as Chaudhry et al. [2019a], Aljundi et al. [2019a]. We thus run Algorithm 4, with an additional online classifier being updated performed at

method	Accuracy (↑)		Forgetting (↓)	
	M=20	M = 50	M=20	M=50
iid online	60.8 ± 1.0	60.8 ± 1.0	N/A	N/A
iid offline	79.2 ± 0.4	79.2 ± 0.4	N/A	N/A
GEM [Lopez-Paz and Ranzato, 2017]	16.8 ± 1.1	17.1 ± 1.0	73.5 ± 1.7	70.7 ± 4.5
iCarl (5 iter) [Rebuffi et al., 2017]	28.6 ± 1.2	33.7 ± 1.6	49 ± 2.4	40.6 ± 1.1
fine-tuning	18.4 ± 0.3	18.4 ± 0.3	85.4 ± 0.7	85.4 ± 0.7
ER	27.5 ± 1.2	33.1 ± 1.7	50.5 ± 2.4	35.4 ± 2.0
ER-MIR [Aljundi et al., 2019a]	29.8 ± 1.1	40.0 ± 1.1	50.2 ± 2.0	30.2 ± 2.3
ER-JPEG	33.9 ± 1.0	43.1 ± 0.6	54.8 ± 1.2	44.3 ± 0.9
Gumbel AE [Riemer et al., 2018]	25.5 ± 2.0	28.8 ± 2.9	71.5 ± 2.8	67.2 ± 3.9
AQM (ours)	$\mathbf{43.5 \pm 0.7}$	$\mathbf{47.0 \pm 0.8}$	$\mathbf{23.0 \pm 1.0}$	$\mathbf{19.0 \pm 1.4}$

Table 5.1 Shared head results on disjoint CIFAR-10. Total memory per class M measured in sample memory size. We report (a) Accuracy, (b) Forgetting (lower is better). Standard error is reported.

line 15. Here we consider the more challenging continual classification setting often referred to as using a *shared-head* [Aljundi et al., 2019a, Farquhar and Gal, 2018, Aljundi et al., 2019c]. Here the model is not informed of the task (and thereby the subset of classes within it) at test time. This is in contrast to other (less realistic) CL classification scenarios where the task, and therefore subset of classes, is provided explicitly to the learner [Farquhar and Gal, 2018, Aljundi et al., 2019a].

For this set of experiments, we report accuracy, i.e. $\frac{1}{T}\sum_{i=1}^{T} R_{T,i}$, and forgetting, i.e. $\frac{1}{T-1}\sum_{i=1}^{T-1} \max(R_{:,i}) - R_{T,i}$ with $R \in \mathbb{R}^{T \times T}$ representing the accuracy matrix where $R_{i,j}$ is the test classification accuracy on task j when task i is completed.

Baselines A basic baseline for continual supervised learning is Experience Replay (**ER**). It consists of storing old data in a buffer to replay old memories. Although relatively simple recent research has shown it is a critical baseline to consider, and in some settings is actually state-of-the-art [Chaudhry et al., 2019a, Aljundi et al., 2019a,

Rolnick et al., 2018]. AQM can be used as an add-on to ER that incorporates online continual compression. We also compare against ER with standard JPEG compression. In addition we consider the following baselines. **iid online** (upper-bound) trains the model with a single-pass through the data on the same set of samples, but sampled iid. **iid offline** (upper-bound) evaluates the model using multiple passes through the data, sampled iid. We use 5 epochs in all the experiments for this baseline. **fine-tuning** trains continuously upon arrival of new tasks without any forgetting avoidance strategy. **iCarl** [Rebuffi et al., 2017] incrementally classifies using a nearest neighbor algorithm, and prevents catastrophic forgetting by using stored samples. **GEM** [Lopez-Paz and Ranzato, 2017] uses stored samples to avoid increasing the loss on previous task through constrained optimization. It has been shown to be a strong baseline in the online setting. It gives similar results to the recent A-GEM Chaudhry et al.. **ER-MIR** [Aljundi et al., 2019a] controls the sampling of the replays to bias sampling towards samples that will be forgotten. We note that the ER-MIR critera is orthogonal to AQM, and both can be applied jointly. **Gumbel AE** Riemer et al. [2018] learns an autoencoder for ER using the Gumbel softmax to obtain discrete representations.

We evaluate with the standard CIFAR-10 split [Aljundi et al., 2019c], where 5 tasks are presented sequentially, each adding two new classes. Evaluations are shown in Table 5.1. Due to our improved storage of previous data, we observe significant improvement over other baselines at various memory sizes. We can contrast AQM's performance with ER's to understand the net impact of our compression scheme. Specifically, AQM improves over ER by 16.0% and 13.9% in the M=20 and M=50 case, highlighting the effectiveness of online compression. Our approach is also superior in forgetting by a significant margin in both memory settings.

To compare directly to reporting in Riemer et al. [2018] we also benchmarked our implementation on the Incremental CIFAR-100 multi-head experiment [Lopez-Paz and Ranzato, 2017] with the same settings as in Riemer et al. [2018]. By using AQM we were

able to get **65.3** vs the reported **43.7** using a buffer of size 200. To specifically isolate the advantage of gumbel softmax versus the vector quantization for drift, we replaced the vector quantization approach with gumbel softmax in an AQM. We observed signficantly less drift in the case where vector quantization is used. Full details of this experiment are described in the supplementary materials along with visualizations.

The CIFAR-10 dataset has a low resolution ($3 \times 32 \times 32$) and uses a lot of data per task (10K samples). These two characteristics might leave the online compression problem easier than in a real-life scenario. Specifically, if the first tasks are long enough and the compression rate is not too large, the model can quickly converge and thus not incur too much representation drift. Indeed, we found that using a single module is already sufficient for this task. For these reasons, we now study the AQM in more challenging settings presented in the next section.

5.3.2 Offline Evaluation on Larger Images

Besides the standard continual classification setup, we propose several other evaluations to determine the effectiveness of the stored data and compression module after learning online compression. We also perform a detailed ablation to study the efficacy of each component in AQM.

Offline training on Imagenet We compare the effectiveness of the stored memories of AQM after a certain amount of online continual compression. We do this by training in a standard iid way an offline classification model using only reconstructions obtained from the storage sampled after online continual compression has progressed for a period of time. In each case we would have the same sized storage available. We note that simply having more stored memories does not amount to better performance as their quality may be severely degraded and affected by drift.

Using this evaluation we first compare a standard reservoir sampling approach on uncompressed data to a 2 module AQM using the same size storage. We observe that performance is drastically increased using the compressed samples. We then use this to perform a series of ablations to demonstrate each component of our proposal is important. Specifically (a) we restrict AQM to have only one module, (b) instead of decoupled training we train modules end-to-end, (c) we remove adaptive compression, thus all samples are stored in the most compressed block, regardless of quality, and (d) we do not stabilize the codebook, the embedding matrices of every block are never fixed. We observe that all these elements contribute to successfully storing a representative set of data for the distribution online.

Method	Accuracy
RS	5.2 ± 0.2
2 Module AQM (ours)	$\mathbf{23.2 \pm 1.1}$
Ablate 2nd Module	20.5 ± 1.3
Ablate Fixing Codebook	19.2 ± 0.6
Ablate Decoupled Training	16.5 ± 0.7
Ablate Adaptive Compression	13.1 ± 3.2

Table 5.2 Imagenet offline training evaluation from online continual compression. We see a clear gain over a standard Reservoir sampling approach. We then ablate each component of our proposal showing each component is important. Note storage used in each experiment is identical (including accounting for model sizes). Standard error is reported.

Drift Ablation We have seen the importance of codebook freezing when dealing with high dimensional datasets. However, judging solely from the final downstream task performance it's difficult to see if the model continues adapting after freezing. As alluded in Sec 5.2.2 there is a tradeoff between keeping recoverable representations and a model's ability to continue to adapt. To shed some light on this, we run the following experiment: we run a vanilla VQ-VAE on the same 20 task mini-imagenet stream, without storing any samples. When it reaches a pre-specified performance

threshold, we fix the codebook, and store compressed *held-out* data from the first task. We then continue to update the VQ-VAE parameters, and the memory is kept fixed for the rest of the stream. We apply self-replay but no other AQM mechanisms (e.g. no sampling from the input stream and no adaptive compression).

We monitor how well the VQ-VAE can adapt by looking at the streaming reconstruction cost, measured on the incoming data before an update. We also monitor the drift of samples stored in the buffer. Results are presented in Figure 5.5. They demonstrate that drift is controlled by stabilizing the codebook, while the model can still improve at nearly the same rate. Further analysis, along with an additional experiment showcasing the robustness of vector quantization to small perturbations is included in the appendix.

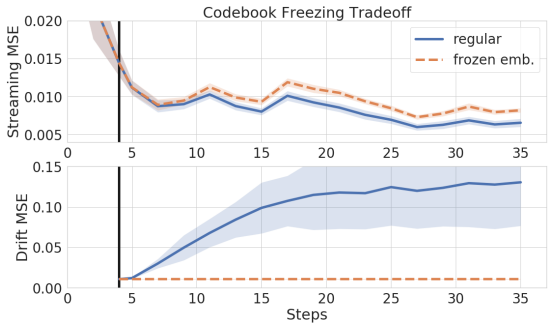

Figure 5.5 Impact of codebook freezing. Vertical black line indicates freezing point. We see that AQM is still able to adapt and reduce its reconstruction loss, while having stable compressed representations. Results averaged over 5 runs, shading represents one standard deviation.

LiDAR Range data enables autonomous vehicles to scan the topography of their surrounding, giving precise measurements of an obstacle's relative location. In its raw form, range data can be very large, making it costly to transmit in real time, or for long term storage. Equipping self-driving cars with a good lidar compressor can enable fast vehicle-to-vehicle (V2V) communication, leading to safer driving Eckelmann [2017]. Moreover, since data collected by autonomous vehicles can be highly non-stationary (new objects on the road, changing weather or traffic conditions), having a compressor which can quickly adapt to this distribution change will reduce the required memory

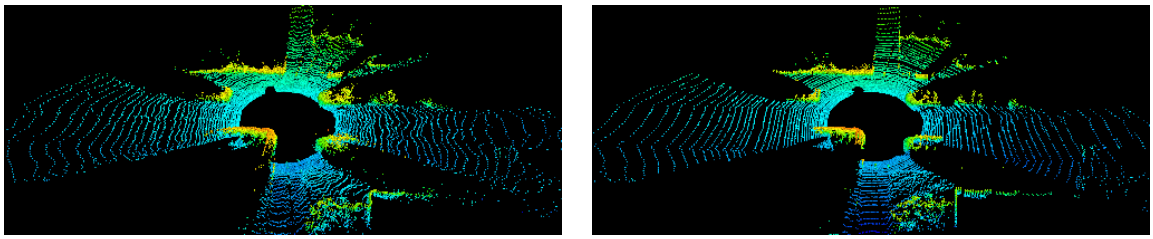

Figure 5.6 Top: Sample decoded from the buffer at the end of training from scratch (32x compression rate). Bottom: Original lidar

for storage (or bandwidth for real time transmission).

We proceed to train AQM on the Kitti Dataset [Geiger et al., 2013], which contains 61 LiDAR scan recordings, each belonging to either the “residential", “road", “city" environments. The data is processed as in Caccia et al. [2019], where points from the same elevation angle are sorted in increasing order of azimuth angle along the same row. This yield a 2D grid, making it compatible with the same architecture used in the previous experiments. As in [Caccia et al., 2019, Tu et al., 2019], we report the reconstruction cost in Symmetric Nearest Neighbor Root Mean Squared Error (SNNRMSE) which allows to compare two point clouds. Note AQM can also be adapted to use task relevant criteria besides `MSE`.

We consider two settings. In the first, we train AQM *from scratch* on a data stream consisting of recordings from all three environments. We present (once) all the recordings of an environment before moving on to another, in order to maximise the distribution shift. We show qualitative results in Figure 5.6 and in the supplementary materials. Observe that we are able to effectively reconstruct the LiDAR samples and can easily tradeoff quality with compression. Overall we obtain **18.8 cm** SNNRMSE with $32\times$ compression, which lies in a range that has been shown in [Tu et al., 2019] to be sufficient to enable SLAM localization with very minimal error. In the second setting, we wish to simulate a scenario where some data is available a priori for the model to leverage. However, this data is limited and does not cover all the possi-

ble modalities to which an autonomous vehicle could be exposed. To this end, we pretrain AQM in a fully offline iid manner on the road and residential recordings. We then simulate the deployment of the compressor on a vehicle, where it must compress and transmit in real time the lidar data feed from a new distribution. We therefore stream the held-out city recordings and show that AQM can be fine-tuned on the fly to reduce the required bandwidth for data transmission. Quantitative results are presented in table 5.3. We ensure that the reconstructed lidar scans have a SNNRMSE smaller than 15.0 cm. Moreover, since the stored representations in AQM are 2D and discrete, we can apply lossless compression schemes such as Portable Network Graphics (PNG).

Method	Size in Mb
Raw	1326.8
Gzip	823.0
AQM	$35.5 \pm .06$
AQM + finetune	$33.0 \pm .07$
AQM + finetune + PNG	$27.9 \pm .01$

Table 5.3 Compression results for the data transmission of the city lidar recordings. We require that each compressed scan has an SNNRMSE under 15 cm. We report standard error.

5.3.3 Atari RL Environments

Another application of online continual compression is for preserving the states of an reinforcement learning agent operating online. These agents may often learn new tasks or enter new rooms thus the observations will often be highly non-iid. Furthermore many existing reinforcement learning algorithms already rely on potentially large replay buffers which can be prohibitive [Mnih et al., 2014, Rolnick et al., 2018] to run and may greatly benefit from an approach such as the AQM to run concurrently with reinforcement learning algorithms. We thus perform a proof of concept for the AQM for storing the state sequence encountered by an RL learner in the atari environment [Bellemare et al., 2013]. We use the dataset and tasks introduced in Anand et al. [2019],

which runs a random or learned policy in the atari environments and provides a set of classification tasks to evaluate whether key information about the state is preserved. Results are shown Table 5.8. We run the online learning with the AQM on the data stream observed by the random agent. We use the same observations and optimization as in Anand et al. [2019] and report the F1 results of a linear probe directly on states for our reconstructions after online compression and the originals. Results for 3 environments are shown in Table 5.8 and examples in in Fig 5.7 and the Appendix. We find that AQM can well preserve the critical information while compressing the state by 16x. The reference accuracies achieved by our classifier are similar to those in Anand et al. [2019]. However, we do not control for the representation size unlike those evaluations of various unsupervised models.

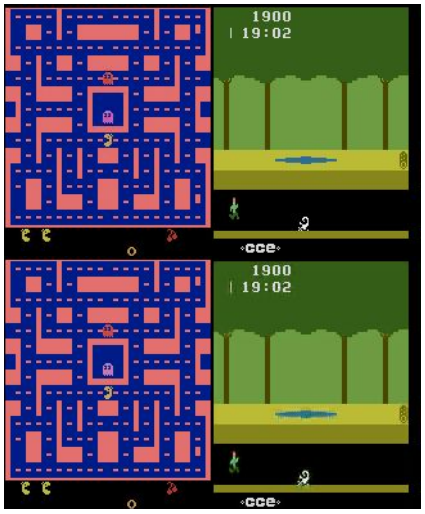

Figure 5.7 Top: original. Bottom: reconstructed from AQM

Game	Cls Input	F1
Pong	Orig. State	86.7
	AQM Recon	86.8
Ms Pacman	Orig. State	89.4
	AQM Recon	88.3
Pitfall	Orig. State	68.2
	AQM Recon	66.7

Figure 5.8 Results on RL probing tasks from Anand et al. [2019] with linear probe applied to original observation and to reconstructions from AQM after online compression. Acc is averaged for each game over game specific prediction.

5.4 Related Work

Learned compression has been recently studied for the case of image compression. For lossy compression, Theis et al. [2017], Ballé et al. [2016], Johnston et al. [2018] have

shown that DNNs trained to compress images can outperform standard algorithms like JPEG. Deep Generative Models can also be used to perform lossless compression. This approach is based on the idea of using a probabilistic model to capture the structure and dependencies in the data, and then using entropy coding to encode the data in a compressed form [Townsend et al., 2019]. However, for both compression settings, these methods are difficult to adapt for online settings as they do not directly address the challenges of the OCC problem (e.g. representation drift).

Continual Learning For a more in-depth survey of CL methods, we refer the reader to chapter 3. Most closely related to the work presented in this chapter, Riemer et al. [2017] consider compressing memories for use in the continual classification task. They also employ a discrete latent variable model but with the Gumbel approximation, which shows to be less effective than our approach. Furthermore a separate offline iid pre-training step for the learned compression is required in order to surpass the ER baseline, distinctly different from the online continual compression we consider.

Lidar compression is considered in Tu et al. [2019] and Caccia et al. [2019]. Both approaches use a similar projection from 3D (x, y, z) coordinates to 2D cylindrical coordinates, and leverage deep generative models to compress the data. However, neither accounts for potential distribution shift, nor for online learning. In this work we show that using this 2D projection in conjunction with our model allows us to mitigate the two issues above for lidar data.

5.5 Discussion

In this chapter, we introduced the problem of online continual compression. We demonstrated that vector quantization can be used to control drift, and we showed how to create mechanisms that allow maintaining compression quality while maximizing mem-

ory usage. These allowed learning compression while compressing, and removed the need for a separate pretraining step. We have shown effectiveness of this online compression approach on standard continual classification benchmarks, as well as for compressing larger images, lidar, and atari data.

The main limitation of the work presented in this chapter is the underlying premise that storage, and not computation, is an important bottleneck when developing continual learning algorithms. With the benefit of hindsight, it has become clear that the field of machine learning is shifting towards large, general-purpose foundation models [Bommasani et al., 2021]. When utilizing such models, the cost of training far surpasses the expenses related to storing the data and model parameters. That being said, I believe the insight developed in this work still has several relevant applications. These applications revolve around AQM's solution to the *representation drift* problem, which is also encountered in other scenarios. Within the field of Continual Learning, the approach of *latent replay* [Pellegrini et al., 2020, Ostapenko et al., 2022, Hayes et al., 2020], where intermediate representations rather than raw data is replayed, is an effective way to reduce the computation cost. Indeed, given a network with L layers, if latent representations are stored for the K-th layer, when performing the rehearsal step, only the last $L - K$ layers are used. This approach, however, suffers from representation drift, as the representations from the first K layers change over time. Hayes et al. [2020] address this by freezing the first K layers, which limits the model's ability to adapt to changes in the data distribution. Pellegrini et al. [2020] also freeze parts of the network, only adapting the batch normalization statistics in the first K layers, and yet *still* observe a performance decrease in later stages, likely attributable to representation drift. The use of a vector quantization bottleneck and codebook freezing could be integrated in latent replay, addressing representation drift without significantly hindering the model's plasticity. Lastly, in self-supervised learning of image representations [Chen et al., 2020b, Caron et al., 2021], contrastive methods like MoCo [He et al., 2020]

compare old encodings with newer ones during training. To ensure that the representations remain consistent (i.e. that there is no significant drift), the older features are obtained from a slow moving encoder, updated via an exponential moving average. The use of quantization as proposed in AQM could be a cheaper alternative to this approach, both in memory and computation.

5.6 Follow-up findings in the community

The role played by compression in rehearsal based CL approaches has been further explored after the publishing of this paper. Wang et al. [2022a] investigate in more detail the trade-off between the reconstruction quality and the amount of samples stored, and further propose an approach to determine the optimal trade-off. Ayub and Wagner [2021] propose an approach to encode and regenerate pseudo-images, retaining the key characterics for a rehearsal-based classifier to maintain a stable performance. Lastly, [Balaji et al., 2020] show that ER can be performant in large scale CL settings using compressed intermediate activations.

Chapter 6

Reducing Abrupt Representation Change in Online Continual Learning

In Chapters 4 and 5, we explored ways to tackle some of the inefficiencies associated with replay approaches. Although these methods demonstrate robust performance in the continual learning scenarios examined thus far, we will illustrate in this chapter that Experience Replay (ER) remains highly unstable at task boundaries. This is a critical point during learning where catastrophic forgetting is observed. Notably, this issue arises regardless of the buffer size and therefore cannot be easily resolved by simply allocating more resources.

Consider the time point in a stream when a new class is introduced after previous classes have been well learned. If we consider the representation being learned, incoming samples from new classes are likely to be dispersed, potentially near and between representations of previous classes, while the representations of previous classes will typically cluster according to their class. Indeed, one might expect minimal changes to the learned representation of the previous classes, while the new class samples are pushed away from the clusters of old class data. However, with a standard ER algorithm [Chaudhry et al., 2019a], we observe that it is the representations of older classes

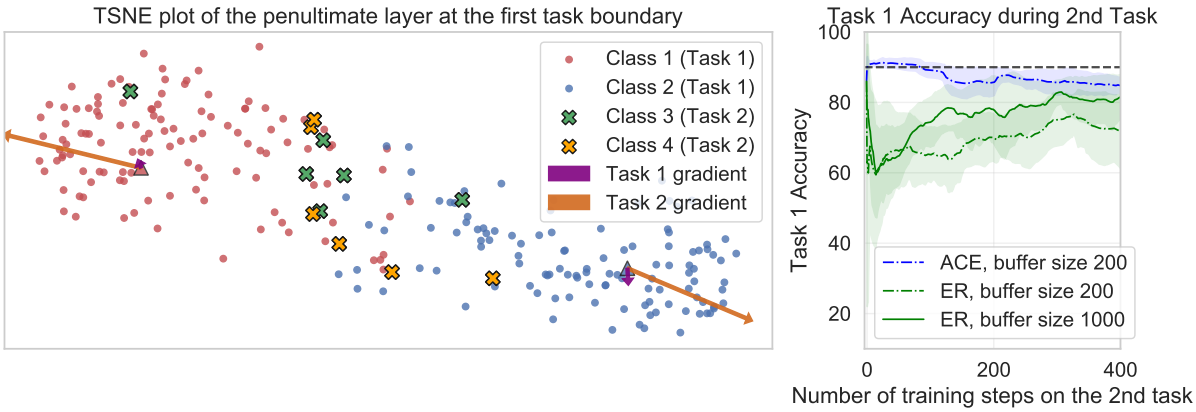

Figure 6.1 (left) Analysis of representations with the first task's class prototypes *at a task boundary*. Under ER when Task 2 begins, class 1 & 2 prototypes experience a large gradient and subsequent displacement caused by the close location of the unobserved sample representations, this leads to a significant drop in performance (right). Our proposed method (ACE) mitigates the representation drift issue and observes no performance decrease on a task switch.

that is heavily perturbed after just a few update steps when training on the new class samples. We hypothesize that the fundamental issue arises from the combination of: new class samples representations lying close to older classes and the loss structure of the standard cross entropy applied on a mix of seen and unseen classes. We illustrate the observed effect in Fig. 6.1 (left).

This behavior is exacerbated especially in the regime of low buffer size. With larger replay buffers, the learner can recover knowledge about the prior classes over time, while with smaller buffers the initial disruptive changes in representations are challenging to correct. Indeed we illustrate this effect in Fig. 6.1 (right), we see that ER only recovers from the initial displacement given a much larger buffer size.

In standard continual learning with replay [Aljundi et al., 2019a, Chaudhry et al., 2019a] the same loss function is usually employed on both the newly received samples and the replayed samples. In contrast, we propose a simple and efficient solution to mitigate this representation drift by using **separate losses on the incoming stream and buffered data**. The key idea is to allow the representations of samples from new classes

to be learned in isolation of the older ones first, by excluding the previously learned classes from the incoming data loss. The discrimination between the new classes and the older ones is learned through the replayed batches, but only after incoming data has been learned, added to the buffer, and made available for replay.

To allow more direct control of the structure in representations we first consider a metric learning based loss for the incoming data, proposed in Khosla et al. [2020], where we propose to exclude samples of previously learned classes from the negative samples. We show that this type of negative selection is critical, and in contrast issues arise when negative examples are sampled uniformly from the buffer. These issues mimic those seen with standard losses in experience replay (ER) [Aljundi et al., 2019a]. On the other hand, we use a different loss on replay buffer data that is allowed to consider new and old classes, thereby consolidating knowledge across current and previous tasks. We call this overall approach ER with *asymmetric metric learning* (ER-AML).

Since cross entropy losses can be more efficient in training for classification than metric learning and contrastive losses (avoiding positive and negative selection) and are widely used in incremental and continual learning, we also propose an alternative cross entropy solution that similarly applies an asymmetric loss between incoming and replay data. Notably, the cross entropy applied to the incoming data *only considers logits of classes of the incoming data*. This variant, named ER with *asymmetric cross-entropy* (ER-ACE), along with ER-AML show strong performance, with little disruption at task boundaries Fig. 6.1 (right). We achieve state-of-the-art results in existing benchmarks while beating different existing methods including the traditional ER solution with an average relative gain of 36% in accuracy. Our improvements are especially high in the small buffer regime. We also show that the mitigation of the old representation drift does not hinder the ability to learn and discriminate the new classes from the old ones. This property emerges from *only learning the incoming data in isolation*; as we will

see, also isolating the rehearsal step (as in Ahn et al. [2021]) leads to poor knowledge acquisition on the current task. Furthermore, we show our ER-ACE objective can be combined with existing methods, leading to additional gains. Finally, we take a closer look at the computation cost of various existing methods. We show that some methods, while obtaining good performance under standard evaluation protocols, fail to meet the computational constraints required in online CL. We provide an extensive evaluation of computational and memory costs across several baselines and metrics.

To summarize, the contributions presented in this chapter are as follows. We first highlight the problem of representation drift in the online continual learning setting. We identify a root cause of this issue through an extensive empirical analysis (Sec. 6.2.2). Second, we propose a new family of methods addressing this issue by treating incoming and past data asymmetrically (Sec. 6.2.1, 6.2.3). Finally, we show strong gains over replay baselines in a new evaluation framework designed to monitor real world constraints (Sec. 6.3). To the best of our knowledge, at the time of the original publication, we were the first to report the computation costs of different methods in our setting, revealing new insights.

6.1 Learning Setting and Notation

We consider the setting where a learner is faced with a possibly never-ending stream of data. At every time step, a labelled set of examples $(\mathbf{X}^{in}, \mathbf{Y}^{in})$ drawn from a distribution D_t is received. However, the distribution D_t itself is sampled at each timestep and can suddenly change to D_{t+1}, when a task switch occurs. The learner is not explicitly told when a task switch happens, nor can it leverage a task identifier during training or evaluation. We note that this definition generalizes task-incremental learning, where each task is seen one after the other. In this scenario, given T tasks to learn, D_t changes $T-1$ times over the full steam, yielding T locally i.i.d learning phases. We also ex-

plore in this section a more general setting without the notion of clearly delineated tasks [Aljundi et al., 2019b, Chen et al., 2020a], where the data distribution gradually changes over time.

Given a model $f_\theta(x)$ representing a neural network architecture with parameters θ, we want to minimize the classification loss $\mathcal{L}$ on the newly arriving data batch while not negatively interfering with the previously learned classes (i.e. increasing the classification loss). A simple and efficient approach to achieve this is to replay stored samples from a fixed size memory, $\mathcal{M}$, in conjunction with the incoming data [Chaudhry et al., 2019a, Rolnick et al., 2018]. The core of our approach is that instead of treating the replayed batch and the incoming one similarly and naively minimizing the same loss, *we opt for a specific loss structure on the incoming batch that would limit the interference with the previously well learned classes.* We approach this by allowing the features of the newly received classes in the incoming data to be initially learned in isolation of the older classes. We first present our idea based on a metric learning loss and then generalize to the widely deployed cross-entropy loss.

6.2 Methods

6.2.1 A Distance Metric Learning Approach for Reducing Drift (ER-AML)

In order to allow fine-grained control of which samples will be pushed away from other samples given an incoming batch, we propose to apply, on the incoming data, a metric learning based loss from Khosla et al. [2020]. Related loss functions have recently popularized in the self supervised learning literature [Chen et al., 2020b]. We combine this in a holistic way with a cross-entropy type loss on the replay data. This allows us to control the representation drift of old classes while maintaining strong classification performance. Note that if a metric learning loss is used alone we need to perform predictions using a Nearest Class Means Rebuffi et al. [2017] approach, which

Algorithm 7 ER-AML

Input: Learning rate α

Initialize: Memory $\mathcal{M}$; Model Params θ

do

  Receive $\mathbf{X}^{in}$ `// Receive from stream`

  $\mathbf{X}_{pos}, \mathbf{X}_{neg} \sim \textsc{FetchPosNeg}(\mathbf{X}^{in}, \mathcal{M})$

  $\mathbf{X}^{bf} \sim \textsc{Sample}(\mathcal{M})$ `// Sample buffer`

  $\mathcal{L} = \gamma \mathcal{L}_1(\mathbf{X}^{in}, \mathbf{X}_{pos}, \mathbf{X}_{neg}) + \mathcal{L}_2(\mathbf{X}^{bf})$

  $SGD(\nabla \mathcal{L}, \theta, \alpha)$ `// Param Update`

  $\textsc{ReservoirUpdate}(\mathcal{M}, \mathbf{X}^{in})$ `// Save`

while *The stream has not ended*

we show is computationally expensive in the online setting.

Given an input data point x, we consider the function $f_\theta(x)$ mapping x to its hidden representation before the final linear projection. We denote the incoming N datapoints by $\mathbf{X}^{in}$ and data replayed from the buffer by $\mathbf{X}^{bf}$. We use the following loss, denoted SupCon [Khosla et al., 2020], on the incoming data $\mathbf{X}^{in}$,

$$\mathcal{L}_1(\mathbf{X}^{in}) = -\sum_{\mathbf{x}_i \in \mathbf{X}_{in}} \frac{1}{|P(\mathbf{x}_i)|} \sum_{\mathbf{x}_p \in P(\mathbf{x}_i)} \log \frac{\texttt{sim}(f_\theta(\mathbf{x}_p), f_\theta(\mathbf{x}_i))}{\sum_{\mathbf{x}_n \in N \cup P(\mathbf{x}_i)} \texttt{sim}(f_\theta(\mathbf{x}_n), f_\theta(\mathbf{x}_i))}, \tag{6.1}$$

where $\texttt{sim}(a, b) = \exp(\frac{a^T b}{\tau \|a\| \|b\|})$ computes the exponential cosine similarity between two vectors, with scaling factor τ [Qi et al., 2018, He et al., 2020].

Here we denote the incoming data $\mathbf{x}_i \in \mathbf{X}^{in}$. We use the P and N to denote the set of positive and negatives with respect to $\mathbf{x}_i$ and the positive examples x_p are selected from the examples in $\mathbf{X}^{in} \cup \mathcal{M}$, which are from the same classes as $\mathbf{x}_i$. In the sequel, we will consider $\mathbf{x}_n$ selected from $\mathbf{X}^{in} \cup \mathcal{M}$ in two distinct ways: (a) from a mix of current and previous classes and (b) only from classes of the $\mathbf{X}^{in}$. Note that this implicitly learns a distance metric where samples of the same class lie close by. For the rehearsal step, we apply a modified cross-entropy objective as per Qi et al. [2018] which allows

us to link the similarity metric from above to the logits.

$$\mathcal{L}_2(\mathbf{X}^{bf}) = -\sum_{x \in \mathbf{X}_{bf}} \log \frac{\texttt{sim}(\mathbf{w}_{c(x)}, f_\theta(\mathbf{x}))}{\sum_{c \in C_{all}} \texttt{sim}(\mathbf{w}_c, f_\theta(\mathbf{x}))} \tag{6.2}$$

where C_{all} the set of all classes observed, and $c(x)$ denotes the label of x. The above formulation allows us to interpret the rows of the final projection $\{\mathbf{w}_c\}_{c \in C_{all}}$ as class prototypes and inference to be performed without need for nearest neighbor search. We combine the loss functions on the incoming and replay data

$$\mathcal{L}(\mathbf{X}^{in} \cup \mathbf{X}^{bf}) = \gamma \mathcal{L}_1(\mathbf{X}^{in}) + \mathcal{L}_2(\mathbf{X}^{bf}) \tag{6.3}$$

We refer to this approach as Experience Replay with Asymmetric Metric Learning (ER-AML). We describe the full rehearsal procedure with ER-AML in Algo 7. Note the buffer may contain samples with the same classes as the incoming data stream. The subroutine `FetchPosNeg` is used to find one positive and negative sample per incoming datapoint in $\mathbf{X}^{in}$, which can reside in either the buffer memory $\mathcal{M}$ or in $\mathbf{X}^{in}$.

6.2.2 Negative Selection Affects Representation Drift

The selection of negatives for the proposed loss $\mathcal{L}$ can heavily influence the representation of previously learned classes and is analogous to the key issues faced in the regular replay methods where cross entropy loss is applied to both incoming and replay data. A typical approach in this loss for classification may be to select the negatives from any other class [Hoffer and Ailon, 2015]. However, this becomes problematic in the continual learning setting as the old samples will be too heavily influenced by the poorly embedded new samples that lie close to the old sample representations. To illustrate what is going on in the feature space, consider the case of a ER-AML's $\mathcal{L}_1$ term, which explicitly controls distances between sample representations. $\mathcal{L}_1$ considers the incoming batch samples (containing new classes) as anchors. As the representations from

these classes haven't been learned, anchors may end up placed near or in-between points from previous classes (analogous to the illustration in Figure 6.1). Since the previous classes samples will be clustered together, if we use them as negatives for the incoming sample anchors, the gradients' magnitude of the positive term will be outweighted by the negative terms coming from the new class samples, similar to what is observed in Figure 6.1.

In this case there is a sharp change in gradients norms of the loss w.r.t. the features of previous classes, as we illustrate in the appendix, which leads to a large change in the representation at the task boundary (and subsequently poor performance). On the other hand, if we use only incoming batch examples as negatives we can avoid this excessive representation drift. We illustrate this in Figure 6.2 by showing the representations drift at the task boundaries for ER-AML when using negative samples from all classes and when using only classes in the incoming batch. In the context of the model under consideration we measure the one iteration representation drift of a sample x as $|f_{\theta^t}(x) - f_{\theta^{t+1}}(x)|$, the output of the network being normalized. We observe that naively applying the proposed loss results in large changes of the learned representation. On the other hand, when allowing only negatives from classes in the incoming batch, we see a reduction in this representation drift. In the appendix we further demonstrate that the accuracy of models trained using ER-AML with only incoming batch negatives can improve the continual learning system performance by a large margin. We emphasize that ER-AML with all negatives and the regular ER method used for online continual learning suffer from a similar issue and

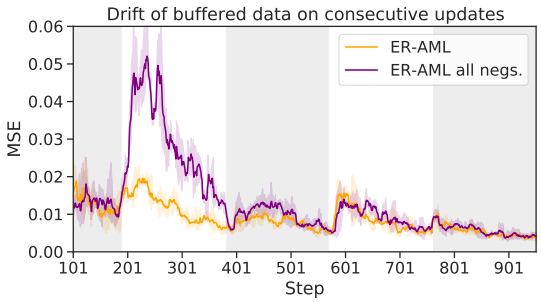

Figure 6.2 Buffer displacement in a 5 task stream. Background shading denotes different tasks.

thus lead to similar poor performances, with appropriate negative selection resolving the problem. This is further emphasized in the appendix , where we observe similar poor drift behavior for ER.

6.2.3 Cross-entropy Based Alternative (ER-ACE)

Having demonstrated the effect of controlling the incoming batch loss in avoiding a drastic representation drift, we now extend it to be applicable to the standard cross-entropy loss typically studied in ER [Aljundi et al., 2019a, Chaudhry et al., 2019a]. Given an incoming data batch, consider C_{old} the set of previously learned classes and C_{curr} the set of classes observed in the current incoming mini-batch. Denoting C the set of classes included in the cross-entropy loss, we define the $\mathcal{L}_{ce}(\mathbf{X}, C)$ cross-entropy loss as: $\mathcal{L}_{ce}(\mathbf{X}, C) = -\sum_{x \in \mathbf{X}} \log \frac{\texttt{sim}(w_{c(x)}, f_\theta(x))}{\sum_{c \in C} \texttt{sim}(w_c, f_\theta(x))}$ where $C \subset C_{all}$ denotes the classes used to compute the denominator. We note that restricting the classes used in the denominator has an analogous effect to restricting the negatives in the contrastive loss. Consider the gradient for a single datapoint x, $\frac{\partial \mathcal{L}_{ce}(x,C)}{\partial f^n_\theta} = \mathbf{W}((\vec{p} - \vec{y}) \odot 1_{\vec{y} \in C})$. Here $\vec{p}$ denotes the softmax output of the network, $\vec{y}$ a one-hot target, $1_{\vec{y} \in C}$ a binary vector masking out classes not in C, and $\mathbf{W}$ the matrix with all class prototypes $\{\mathbf{w}_c\}_{c \in C_{all}}$. When the loss is applied in the batch setting, it follows that only prototypes whose labels are in C will serve roles analogous to positives and negatives in the contrastive loss. We can then achieve a similar control as the metric learning approach on the learned representations.

Now, our loss applied at each step would be:

$$\mathcal{L}_{ace}(\mathbf{X}^{bf} \cup \mathbf{X}^{in}) = \mathcal{L}_{ce}(\mathbf{X}^{bf},\ C_{old} \cup C_{curr}) + \mathcal{L}_{ce}(\mathbf{X}^{in}, C_{curr})$$

where C_{curr} denotes the set of the classes represented in the incoming batch and C_{old} denotes previously seen classes that are not presented in the incoming batch, those that

we want to preserve their representation. Note that this is a straightforward procedure and induces no additional computational overhead. We refer to it as Experience Replay with Asymmetric Cross-Entropy (ER-ACE).

6.3 Experiments

We have highlighted the issue of abrupt representation change when new classes are introduced, and propose two methods that address this issue. We now demonstrate that mitigating drift directly leads to better performance on standard online continual learning benchmarks. As in Lopez-Paz and Ranzato [2017], Aljundi et al. [2019a], Chaudhry et al. [2019a] we use a reduced Resnet-18 for our experiments, and leave the *batch size* and the *rehearsal batch size* fixed at *10*. This allows us to fairly compare different approaches, as these parameters have a direct impact on the computational cost of processing a given stream.

6.3.1 Datasets

All benchmarks are evaluated in the single-head setting, i.e. task descriptors are not provided to the model at test time, hence the model performs N-way classification where N is the total amount of classes seen.

Split CIFAR-10 partitions the dataset into 5 disjoint tasks containing two classes each (as in Aljundi et al. [2019a], Shim et al. [2021])

Split CIFAR-100 comprises 20 tasks, each containing a disjoint set of 5 labels. We follow the split in Chaudhry et al. [2019a]. All CIFAR experiments process 32×32 images.

Split MiniImagenet splits the MiniImagenet dataset into 20 disjoint tasks of 5 labels each. Images are 84×84.

6.3.2 Baselines

We focus our evaluation on replay-based methods, as they have been shown to outperform other approaches in the online continual learning setting Chaudhry et al. [2019a], Aljundi et al. [2019a], Ji et al. [2020]. We keep buffer management constant across methods : all samples are kept or discarded according to Reservoir Sampling Vitter [1985]. We consider the following state-of-the-art baselines:

ER: Experience Replay with a buffer of a fixed size. Unlike Aljundi et al. [2019a], we do not leverage the task identifier during training to ensure that rehearsal samples belong to previous classes.

iCaRL Rebuffi et al. [2017] A distillation loss alongside binary cross-entropy is used during training. Samples are classified based on closest class prototypes, obtained from recomputing and averaging buffered data representations.

MIR Aljundi et al. [2019a] selects for replay samples interfering the most with the incoming data batch.

DER++ Buzzega et al. [2020] uses a distillation loss on the logits to ensure consistency over time.

SS-IL Ahn et al. [2021] learns both the current task loss and the replay loss in isolation of each other. An additional task-specific distillation is used on the rehearsal data.

GDUMB Prabhu et al. [2020] performs offline training on the buffer with unlimited computation and unrestricted use of data augmentation at the end of the task sequence.

iid: The learner is trained with a single pass on the data, in a single task containing all the classes. We also consider a version of this baseline using a similar compute budget as replay methods (**iid++**)

We note additional baselines such as Lopez-Paz and Ranzato [2017], Chaudhry et al.

were shown to perform poorly in this setting by prior work Buzzega et al. [2020] and are thus left out for clarity.

6.3.3 Evaluation Metrics and Considerations

Our evaluation includes the metrics and experimental settings used in previous works on online continual learning with a single-head [Aljundi et al., 2019a, Ji et al., 2020, Shim et al., 2021]. We provide extra emphasis on anytime evaluation and comparisons of the computation time per incoming batch. We also consider several additional settings in terms of computation and use of image priors.

Anytime evaluation A critical component of online learning is the *ability to use the learner at any point* De Lange et al. [2019]. Although most works in the online (one-pass through the data) setting report results throughout the stream [Lopez-Paz and Ranzato, 2017, Chaudhry et al., Aljundi et al., 2019c], several prior works have reported the final accuracy as a proxy [Aljundi et al., 2019a, Shim et al., 2021]. However, a lack of anytime evaluation opens the possibility to exploit the metrics by proposing offline learning baselines that are inherently incompatible with anytime evaluation [Prabhu et al., 2020].

In order to make sure that learners are indeed online learners, we evaluate them throughout the stream. We define the Anytime Accuracy at time k (AA_k) as the average accuracy on the test sets of all distributions seen up to time k. If the learning experience lasts T steps, then AA_T is equivalent to the final accuracy. Finally, we report the *Averaged Anytime Accuracy (AAA)* [Caccia et al., 2020b], which measures how well the model performed over the learning experience

$$\text{AAA} = \frac{1}{T}\sum_{t=1}^{T}(AA)_t. \tag{6.4}$$

Computation and Memory Constraints While memory constraints are well documented in previous work, careful monitoring of computation is often overlooked; some methods can indeed hide considerable overhead which can make the comparison across methods unfair. On the other hand, this is critical to the use cases of online continual learning. To remedy this, we report for each method the total number of FLOPs used for training. While we cannot fix this quantity as we can for memory (since different methods require different computations), this will shed some light on how different methods compare. Note that we also include in this total any inference overhead required by the models; Nearest Class Mean (NCM) classifiers must compute class prototypes before inference for example. We add this cost every time the model is queried to measure its Anytime Accuracy. Let

$$\text{Mem} = \frac{1}{T}\sum_{t=1}^{T}|\theta_t|+|\mathcal{M}_t|, \quad \text{Comp} = \sum_{t=1}^{T}\mathcal{O}(m(\cdot;\theta_t)), \tag{6.5}$$

where $\mathcal{O}(m(\cdot;\theta_t))$ and $|\mathcal{M}_t|$ denote the number of FLOPs and the size of the buffer used at time t. Since the same backbone and buffer is used for all methods in this chapter, we will focus our constraint analysis on computation.

Data Augmentation In the settings of Aljundi et al. [2019a], Lopez-Paz and Ranzato [2017], Ji et al. [2020], Shim et al. [2021], Chaudhry et al. [2019a] data augmentation is not used. However, this is a standard practice for improving the performance on small datasets and can thus naturally complement most methods utilizing replay buffers. Notably, Prabhu et al. [2020], the offline learning method, utilizes data augmentation when comparing to the above online learners. To avoid unfair comparisons, in our experiments we indicate when a method uses augmentation. When not specified, we treat it as a hyperparameter and report the best performance.

Hyperparameter selection For all datasets considered, we withhold 5 % of the training data for validation. For each method, optimal hyperparameters were selected via a grid search performed on a validation set. The selection process was done on a per dataset basis, that is we picked the configuration which maximized the accuracy averaged over different memory settings. We found that for both ER-AML and ER-ACE, the same hyperparameter configuration worked across all settings and datasets. All necessary details to reproduce our experiments can be found in the appendix.

6.3.4 Standard Online Continual Learning Settings

We evaluate on Split CIFAR-10, Split CIFAR-100 and Split MiniImagenet using the protocol and constraints from Aljundi et al. [2019a], Ji et al. [2020], Shim et al. [2021] . We note in all results each method is run 10 times, and we report the mean and standard error. We first discuss dataset specific results, before analysing the computation cost of each method.

CIFAR-10 results are found in Table 6.1 using a variety of buffer sizes. In this setting, we see that both the methods we propose, ER-AML and ER-ACE *consistently outperform* other methods by a significant margin. This result holds in both settings where data augmentation is (or not) used, outperforming previous state-of-the-art methods MIR and DER++. Shifting our attention to SS-IL, its underperformance w.r.t to ER-ACE highlights the importance of having a rehearsal objective that considers the new classes. In the appendix , we observe that when applying SS-IL in the online setting: (1) the method performs poorly on the current task, as is it unable to consolidate old and new knowledge, (2) yet mitigates representation drift even on a perfectly balanced stream. The latter is surprising, as the method was designed specifically to address stream imbalance. Finally, we note the offline training baseline G-DUMB cannot satisfy the anytime evaluation criteria.

Method	Data Aug.	$M = 5$ AAA	$M = 5$ Acc	$M = 20$ AAA	$M = 20$ Acc	$M = 100$ AAA	$M = 100$ Acc	Train TFLOPs	Mem. (Mb)
iid		-	$62.7_{\pm 0.7}$	-	$62.7_{\pm 0.7}$	-	$62.7_{\pm 0.7}$	8	4
iid++		-	$72.9_{\pm 0.7}$	-	$72.9_{\pm 0.7}$	-	$72.9_{\pm 0.7}$	16	4
DER++	✓	$50.7_{\pm 1.1}$	$31.8_{\pm 0.9}$	$55.6_{\pm 1.2}$	$39.3_{\pm 1.0}$	$60.1_{\pm 1.3}$	$52.3_{\pm 1.1}$	24	**(4, 7)**
ER		$40.0_{\pm 0.8}$	$19.7_{\pm 0.3}$	$45.2_{\pm 1.3}$	$26.7_{\pm 1.0}$	$55.4_{\pm 1.4}$	$38.7_{\pm 0.8}$	**17**	**(4, 7)**
	✓	$45.6_{\pm 1.1}$	$28.4_{\pm 1.0}$	$55.9_{\pm 1.2}$	$40.3_{\pm 0.6}$	$60.3_{\pm 1.3}$	$49.4_{\pm 1.3}$		
iCaRL†		$47.0_{\pm 0.8}$	$30.6_{\pm 0.8}$	$55.1_{\pm 0.7}$	$41.7_{\pm 0.6}$	$59.3_{\pm 0.6}$	$45.1_{\pm 0.6}$	(21, 47)	(8, 11)
	✓	$49.1_{\pm 1.0}$	$33.4_{\pm 1.0}$	$54.4_{\pm 0.7}$	$39.2_{\pm 0.8}$	$56.9_{\pm 0.7}$	$42.3_{\pm 0.8}$		
MIR†		$39.3_{\pm 1.0}$	$19.7_{\pm 0.5}$	$44.7_{\pm 1.1}$	$29.7_{\pm 0.6}$	$53.8_{\pm 1.7}$	$43.3_{\pm 1.0}$	41	**(4, 7)**
	✓	$44.9_{\pm 0.9}$	$29.8_{\pm 0.8}$	$49.7_{\pm 1.0}$	$41.8_{\pm 0.6}$	$54.6_{\pm 1.4}$	$49.3_{\pm 0.6}$		
SS-IL†		$42.6_{\pm 1.7}$	$29.6_{\pm 0.4}$	$44.8_{\pm 1.8}$	$35.1_{\pm 0.9}$	$48.1_{\pm 2.2}$	$41.1_{\pm 0.4}$	19	(8, 11)
	✓	$41.1_{\pm 1.6}$	$31.6_{\pm 0.5}$	$47.0_{\pm 1.2}$	$38.3_{\pm 0.4}$	$48.1_{\pm 1.7}$	$47.5_{\pm 0.7}$		
ER-ACE (ours)		**53.1**$_{\pm 1.0}$	**35.6**$_{\pm 1.0}$	**58.0**$_{\pm 0.7}$	$42.6_{\pm 0.7}$	**61.9**$_{\pm 0.9}$	$52.2_{\pm 0.7}$	**17**	**(4, 7)**
	✓	**52.6**$_{\pm 0.9}$	**35.1**$_{\pm 0.8}$	**56.4**$_{\pm 1.0}$	$43.4_{\pm 1.6}$	**61.7**$_{\pm 0.9}$	**53.7**$_{\pm 1.1}$		
ER-AML (ours)		$49.4_{\pm 1.0}$	$30.9_{\pm 0.8}$	**57.0**$_{\pm 1.0}$	$39.2_{\pm 1.0}$	**63.3**$_{\pm 1.0}$	$52.2_{\pm 1.1}$	**17**	**(4, 7)**
	✓	$50.4_{\pm 1.3}$	**36.4**$_{\pm 1.4}$	**56.8**$_{\pm 1.0}$	**47.7**$_{\pm 0.7}$	**62.0**$_{\pm 0.9}$	**55.7**$_{\pm 1.3}$		
GDUMB	✓	$0_{\pm 0.0}$	$35.0_{\pm 0.6}$	$0_{\pm 0.0}$	$45.8_{\pm 0.9}$	$0_{\pm 0.0}$	$61.3_{\pm 1.7}$	(43, 853)	(11, 14)

Table 6.1 split CIFAR-10 results. † indicates the method is leveraging a task identifier at training time. For methods whose compute depend on the buffer size, we report min and max values. We evaluate the models every 10 updates. Results within error margin of the best result are bolded, we report standard error.

Method	AAA	Acc.	Train TFLOPs	Mem. (Mb.)	AAA	Acc.	Train TFLOPs	Mem. (Mb.)
iid	-	$19.8_{\pm 0.3}$	9	4	-	$16.7_{\pm 0.5}$	59	4
iid++	-	$28.3_{\pm 0.3}$	17	4	-	$25.0_{\pm 0.8}$	118	4
DER++	$23.3_{\pm 0.5}$	$15.1_{\pm 0.4}$	25	36	$21.7_{\pm 0.6}$	$12.9_{\pm 0.3}$	176	217
ER	$24.2_{\pm 0.6}$	$19.8_{\pm 0.4}$	**17**	**35**	$26.2_{\pm 0.8}$	$18.2_{\pm 0.5}$	**118**	**216**
iCaRL†	$26.3_{\pm 0.3}$	$17.3_{\pm 0.2}$	294	39	$24.4_{\pm 0.4}$	$17.1_{\pm 0.1}$	2097	220
MIR†	$23.6_{\pm 0.8}$	$20.6_{\pm 0.5}$	41	**35**	$27.2_{\pm 0.7}$	$20.2_{\pm 0.8}$	294	**216**
SS-IL†	$31.5_{\pm 0.5}$	$25.0_{\pm 0.3}$	19	39	**29.7**$_{\pm 0.6}$	**23.5**$_{\pm 0.5}$	137	220
ER-ACE (ours)	**32.7**$_{\pm 0.5}$	**25.8**$_{\pm 0.4}$	**17**	**35**	**30.2**$_{\pm 0.6}$	**22.7**$_{\pm 0.6}$	**118**	**216**
ER-AML (ours)	$30.2_{\pm 0.6}$	$24.3_{\pm 0.4}$	28	**35**	$27.0_{\pm 0.7}$	$19.3_{\pm 0.6}$	200	**216**

Table 6.2 Split CIFAR-100 (left) and Mini-Imagenet (right) results with $M = 100$. For each method, we report the best result between using (or not) data augmentations. Standard error is reported.

Longer Task Sequence results are shown in Table 6.2 with CIFAR-100 on the left and MiniImagenet on the right. On both datasets similar findings are observed, our proposed methods match or outperform strong existing baselines. SS-IL performs similarly to our method on mini-imagenet hile having a higher computational and memory cost. As mentioned above, the method struggles to learn the current task, however here the “weight" of the current task is small in the final acc. of the 20-task regime. We see that average anytime accuracy is higher for ER-ACE and indeed the anytime curves in the appendix further illustrate this. Finally, ER-ACE shows relative gains of **35%** in accuracy over ER, without any additional computation cost. For Mini-Imagenet, ER-ACE outperforms the single-pass iid baseline, and nearly reaches the performance of the equal-compute iid baseline.

Computation Budget To provide another view of the computational advantages of our proposal, we report the accuracy given compute budget over the length of the sequence in Fig 6.3. When monitoring the computation performed by each baseline, we notice that several methods do not compete on equal footing. First, the use of Nearest Class Mean (NCM) classifiers leads to a significant compute cost, as shown for iCaRL. For our experiments, we evaluate the model after 10 mini-batches (100 total samples), where NCM classifier **must forward the whole buffer** to get class prototypes. We argue that such an approach has disadvantages in the online setting due to poor computational trade-offs. Second, MIR Aljundi et al. [2019a] has an ex-

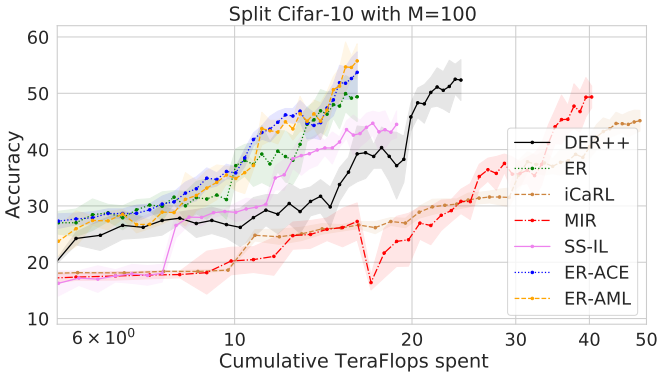

Figure 6.3 Total Accuracy as a function of TeraFLOPs spent. Here the models are evaluated on all 10 classes, to ensure consistency across timesteps.

pensive sample retrieval cost. It remains to be shown if this step can be approximated more efficiently. Finally, we note that our method, ER-AML has varying compute: for streams with a small number of classes per task (CIFAR10), it can compute the incoming loss leveraging only the incoming data. In other datasets, where an incoming batch may not have at least two samples of each class, an additional cost to forward a buffered point is incurred.

Evaluation with augmentation The use of augmentations also permits extra benefits of replay methods particularly in settings where buffer overfitting is more present, e.g. in the small buffer regime. From the results in Table 6.1, we see that *augmentations provides significant gains* for a large set of methods. It is therefore crucial to compare methods on equal footing, where they can all leverage (or not) data augmentation. For example, *gains reported in Prabhu et al. [2020] over ER completely vanish* when ER is given the same access to augmented data. We note that for Mini-Imagenet, augmentations did not help. We hypothesize that since this is the hardest task the risk of overfitting on the buffer is less severe.

6.3.5 Blurry Task Boundaries

Table 6.3 CIFAR-10 Blurry Task Boundary Experiments

Method	$M = 20$	$M = 100$
ER	$32.1_{\pm 1.5}$	$42.7_{\pm 2.2}$
DER++	$31.0_{\pm 1.4}$	$41.7_{\pm 1.4}$
ER-AML	$\mathbf{45.6}_{\pm 1.2}$	$\mathbf{55.2}_{\pm 1.1}$
ER-ACE	$\mathbf{44.5}_{\pm 0.5}$	$50.2_{\pm 1.1}$

Next, we explore a setting where the distribution is continuously evolving, rather than clearly delineated by task boundaries (similar to settings considered in Aljundi et al. [2019c]). To do this, we linearly interpolate between tasks over time, resulting in new classes being slowly mixed into the data stream. This experiment is done on Split-CIFAR10, and the interpolation is such that at every timestep, the incoming data batch has on average 2 unique labels (as in the original experiment). We only evaluate task-free

methods in this setting: methods like MIR and SS-IL cannot be used in such setting. Results in table 6.3 report the final accuracy, averaged over 5 runs, we report the standard error. We observe our ER-AML and ER-ACE methods perform well in this setting. More details provided in the appendix.

6.4 Related Work

For a more complete picture of the standard methods in Continual Learning, we refer the reader to chapter 3. In this section, we focus on work in offline CL designed to address the overestimation of recent classes in the last layer, as these methods share similarities with the ones proposed in this chapter.

6.4.1 Class imbalance in Continual Learning

In this chapter, we investigated the underlying causes for performance degradation in replay-based methods. Related to this study are works in the class incremental setting, where similar to our case a shared output layer is used, *but classes are learned offline*. Works in this area address the implicit class imbalance issue occurring when new classes are learned alongside replayed data. Zhao et al. [2019] propose to correct last layer weights after a group of classes is learned via adjusting the weights norm. Wu et al. [2019] suggest deploying additional parameters in order to linearly correct the "bias" in the shared output layer. Those parameters are learned at the end of each training phase. Hou et al. [2019] consider addressing this imbalance through applying cosine similarity based loss as opposed to the typical cross entropy loss along with a distillation loss and a margin based loss with negatives mining to preserve the feature of previous classes. Recently, Ahn et al. [2021] propose to learn the incoming tasks and the previous tasks separately. They use a masked softmax loss for the incoming and rehearsal data, to counter the class imbalance. All the methods highlighted

above operate in the *offline* setting, where data from the current task can be revisited as needed, making the disruptive issues emphasized at the task boundary less critical. In this chapter, we focus on the online setting, with potentially overlapping tasks. As we show in the appendix, work by Ahn et al. [2021] developed to counter class imbalance can inhibit learning of the current task in the online setting. Lastly, Zeno et al. [2018] uses a logit masking related to our method, but their context is based on the multi-head setting, and does not consider replay based methods, where learning across tasks occurs. Their goal is to activate only the head of which the samples within the new batch belong to. However, our approach is more general, and it applies to the single head setting (where we have a single output layer for all classes, and no task oracle.)

6.5 Discussion

In this chapter, we have demonstrated how the standard loss function in online continual learning settings places excessive pressure on representations of previous classes. We introduced two modifications to the loss function, both of which involve treating incoming and replay data asymmetrically. Our suggested approach does not necessitate knowledge of the current task and has been proven effective in handling long task sequences, delivering high performance with minimal or no additional cost.

Furthermore, we have raised the bar for high-quality evaluation in online continual learning by examining a wide array of baselines and metrics. For the first time in this thesis, we conducted a comprehensive analysis of the computational cost of various methods. Interestingly, we discovered that several approaches previously compared to Experience Replay on equal footing actually have a significantly larger computational footprint. Similarly, we emphasized the importance of anytime evaluation for accurately assessing the performance of methods in an online setting. Relying solely on final average performance fails to distinguish between inherently offline approaches

like GDUMB [Prabhu et al., 2020] and practical online solutions. In short, proper evaluation of methods is critical for a clear understanding of how different approaches compare.

In conclusion, this chapter's findings indicate that abrupt representation drift is the primary cause of catastrophic forgetting in online class-incremental learning settings. It is important to note that this phenomenon is highly specific to the benchmarks and settings used in the previous chapters, and its relevance in realistic CL scenarios remains uncertain. In fact, it has been shown that pretrained model representations are more stable than those initialized randomly [Ramasesh et al., 2022]. Moreover, other tasks, such as generative modeling, may not be affected by a similar phenomenon. In the case of online compression discussed in chapter 5, forgetting in the auto-encoder was gradual rather than "catastrophic." A similar observation was made in Masarczyk et al. [2021], indicating that discriminative models are more susceptible to catastrophic forgetting. This growing body of evidence suggests that benchmarks and metrics used in Continual Learning research may not accurately reflect real-world CL use-cases. In the following chapters, we will deviate from standard evaluation protocols and attempt to identify and address actual challenges faced by practitioners.

6.6 Follow-up findings in the community

The initial motivation which led to the work presented in this chapter was the instability phase seen at task boundaries (Fig. 6.1). A year later, work in Lange et al. [2023] have confirmed our initial findings, and named this phenomenon the "stability gap". Again, it was argued that anytime evaluation is required to properly access performance in CL, so that instability around task boundaries can be detected and addressed. The authors extended our findings, showing that regularization and distillation methods also suffer from instability at task boundaries. Additionally, the authors observed a

similar behavior in domain incremental learning, where $p(y)$ is fixed but $p(x|y)$ changes over time. Thankfully, research in CL is starting to use anytime metrics, and already we have seen work focusing on directly addressing the stability gap [Harun and Kanan, 2023].

In follow-up work, I along with co-authors at MILA have investigated whether a similar phenomenon occurs in Federated Learning (FL). FL is a machine learning approach where multiple devices collaboratively train a model while keeping their data locally, thus improving privacy and reducing the need for data centralization. We discovered in Legate et al. that when the data distribution is highly heterogeneous across clients, a similar behavior to representation shift occurs, and that a modified cross-entropy loss can mitigate this problem, leading to better and faster convergence.

Chapter 7

On Anytime Learning at Macroscale

So far, the earlier chapters have primarily focused on online learning scenarios where the data distribution evolved over time, typically through the introduction of new classes to be learned and discriminated. In these benchmarks, memory consumption was carefully monitored, restricting the size of the replay method's memory buffer, while less attention was given to the computational costs. This chapter takes a different approach, centering the evaluation of methods around their computational footprint. More importantly, we investigate the question of *how to best allocate* compute resources in sequential learning settings, as data is made available over time.

Indeed, in many practical applications of machine learning, data is not static but arrives sequentially in large chunks (or mega-batches). For instance, deployed language modeling systems need to be updated every few months to accommodate new snapshots of the Common Crawl dataset[1]. Similarly, visual object recognition systems need to be updated as new labeled data is gathered thanks to users interacting with the system. Moreover, as computing clusters are equipped with more memory and compute, machine learning practitioners would like to train bigger and bigger models

[1] https://commoncrawl.org/the-data/

on the ever increasing amount of data, since bigger models are often more accurate. In this setting, they face a dilemma: How to maximize the performance of the system at any point in time while satisfying a certain computational budget?

This question has certainly been studied before, most notably in the online learning literature [Cesa-Bianchi and Lugosi, 2006]. For instance, in a contextual bandit setting the learner observes one example at the time and receives a reward after making a prediction. Of course, this can be extended to the case where the input is not just a single example but a set of examples (hereinafter referred to as mega-batch).

While prior works on online learning set a sound theoretical framework, there are some subtle issues that make it not quite applicable to the practical setting described above. First, computation is seldom explicitly taken into account, while in practice algorithms that are too computationally intensive cannot be considered at scale. Second, the vast majority of these works assumes linearity of predictors and convexity of optimization problems, whereby the order of examples does not change the optimum solution. Instead, in many practical applications (like language modeling) we are interested in using deep neural networks which are highly non-linear and which map to non-convex optimization problems. The lack of linearity hinders theoretical analysis, and it has profound practical implications. For instance, according to online learning theory the best case scenario is achieved when there are no delays [Joulani et al., 2016, Flaspohler et al., 2021], meaning that examples and their error signal are best to be consumed right away without any staleness in the model parameters. To use the language of the practitioner training a language model, this means that according to convex online learning the best strategy is to train one mega-batch at the time. This however might not be a good strategy.

Consider what would happen if the deep neural network does multiple passes over each mega-batch before processing the next, and compare its performance to the one of a learner that waits for all the mega-batches to arrive before shuffling all data and

applying the same stochastic gradient descent optimization algorithm as shown on the right part of Fig. 7.1. The latter setting is the standard procedure used in supervised learning (green curve): the learning algorithm optimizes a fixed objective (i.e the empirical risk over the entire training dataset) that is known to produce good predictors. While this predictor obtains the best final performance, it also attains the worst anytime performance since its predictions were random throughout the learning experience. In the former setting, by updating after each new mega-batch (purple curve), we can expect to maintain a good predictor all along the training experience, overcoming the problem described previously. However in this case, the learner is facing a changing learning objective, since each new mega-batch defines a slightly different empirical risk [Jothimurugesan et al., 2018]. While we can expect this effect to be negligible when using linear models which eventually will converge to the same global optimum when all mega-batches are available, this is not the case when using non-linear predictors like deep neural networks. In that case, the sequence of optimization problems generated by the sequence of mega-batches may lead the learner to a completely different (local) optimum than the supervised learning setting, and thus to a completely different predictor. There is thus an open question about how different models behave when performing sequential learning over a stream of mega-batches.

In this chapter, we empirically analyze several deep learning models (§7.2) under the assumption that data comes as a sequence of mega-batches, all drawn from the same distribution for simplicity. Since we are interested in models that attain good performance at any point in time and since we evaluate only after learning on each mega-batch but not during the learning of each individual mega-batch, we dub this learning setting Anytime Learning at MAcroscale (ALMA) (§7.1).

Through extensive empirical analysis (§7.3) we provide supporting evidence that waiting for a few mega-batches before updating the model is often the best strategy, although how long to wait depends on several factors such as the time horizon and

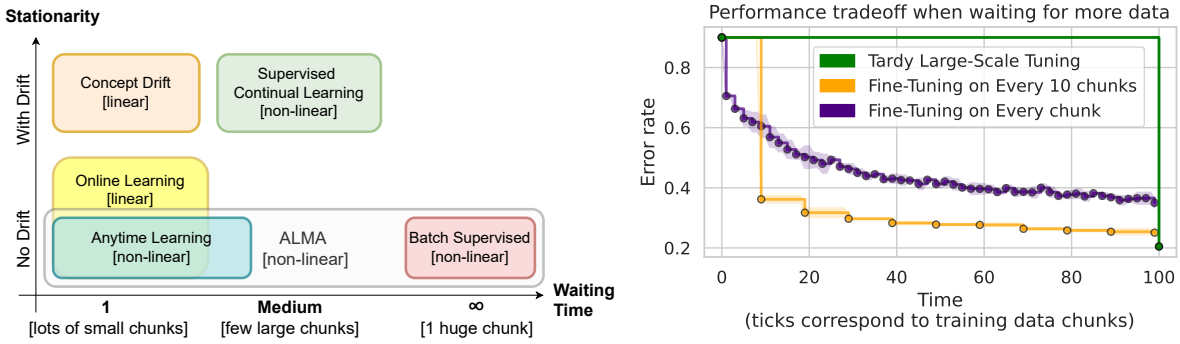

Figure 7.1 Left: ALMA compared to other learning frameworks. In ALMA, mega-batches of data are drawn from the same distribution (no drift) and arrive sequentially, but the learner can decide how long to wait before training on them. In the limit, if the learner waits till the end of the stream then learning reduces to standard batch supervised learning. Right: Examples of CIFAR 10 learning curves varying how long to wait before updating the model. Waiting for a small number of mega-batches before updating the parameters results in lower anytime error rate (smaller area under the learning curve).

model size relative to the amount of data in each mega-batch. Second, bigger models are more statistically efficient and generalize better. Third, none of the approaches we tried for growing the architecture were more effective than simpler alternatives which used fixed architectures, like ensembling. Overall, this study provides clear directions of future research, and also a platform for benchmarking new approaches against well tuned baselines.

7.1 Learning Setting

In anytime learning at macroscale (ALMA), we assume that there exists an underlying data distribution $p(x, y)$ with input $x \in \mathbb{R}^D$ and desired label $y \in \{1, \ldots, C\}$. For the sake of simplicity of exposition, in this work we restrict ourselves to classification problems, but similar arguments can be made for regression. The key property of ALMA is that data is presented to the learner as a stream $\mathcal{S}_B$ of B consecutive batches of examples. Let $\mathcal{D}_i$ be a collection of $N \gg 0$ i.i.d. samples randomly drawn from $p(x, y)$, for $i \in$

$\{1, \ldots, B\}$. The stream is then defined as the ordered sequence $\mathcal{S}_B = \{\mathcal{D}_1, \ldots, \mathcal{D}_B\}$. We refer to each dataset $\mathcal{D}_i$ as *mega-batch*, as it is composed by a large number of examples.

Typically a learner $m : \mathbb{R}^D \rightarrow \{1, \ldots, C\}$ updates its parameters by processing a *mini-batch* of $n \ll N$ examples at the time from each mega-batch $\mathcal{D}_i$ in such a way to minimize its objective function. Since the data is observed as a stream of mega-batches, the learner cannot have access to future mega-batches, and cross-validation of model hyper-parameters can only be performed using a subset of the current mega-batch. In other words, the learner can only do one pass over the stream. However, the learner typically does multiple passes over the current mega-batch if this improves its generalization ability. In fact, the learner might make several passes over the current and some previous mega-batches, although replaying too much might eventually deplete its computational budget.

Either way, since the learner makes several passes over each mega-batch, the overall data distribution observed by the learner by the end of the stream is not i.i.d., even though mega-batches are drawn from the same underlying distribution $p(x, y)$ and samples drawn from each mega-batch are i.i.d.. For instance, in the limit case where each mega-batch consists of a single example from a set of n examples and a learner performing k passes over each mega-batch, the stream will consist of a sequence of examples (in a certain order) each replicated k times, which is different from drawing uniformly at random $k * n$ examples from the original set of n examples. This implies a trade-off between fitting the current data well versus generalizing well by the end of the stream.

In ALMA, the learner has an additional hyper-parameter compared to other learning frameworks: It can decide how long to wait before updating its parameters. We measure such *waiting time* in terms of number of consecutive mega-batches. For instance, a model with a waiting time equal to k, aggregates k consecutive mega-batches before updating its parameters. This will sacrifice a bit its performance during the

waiting period, but might ultimately yield better generalization since the model can better shuffle the data and get closer to the ideal i.i.d. data distribution required by stochastic gradient descent optimization.

7.1.1 Metrics

We evaluate learners in the ALMA setting across three axes, namely: error rate, memory and computation. Let t be the time at which the t-th mega-batch arrives; this data can be used by the model to update its parameters or it is simply aggregated to previous mega-batches for later use.

We compute the error rate of model m at time t (after the arrival and potential update over the t-th mega-batch) and compute the area under the curve obtained varying t from 0 till the total number of mega-batches B; the resulting cumulative error rate (CER) is:

$$\text{CER} = \sum_{t=1}^{B} \frac{1}{|\mathcal{D}^{\text{Ts}}|} \sum_{(x,y)\in\mathcal{D}^{\text{Ts}}} |m(x;\theta_t) \neq y|, \tag{7.1}$$

where $m(x;\theta_t)$ is the model at time t equipped with parameters θ_t, $\mathcal{D}^{\text{Ts}}$ is the test set (common for all mega-batches in the stream), $|\mathcal{D}^{\text{Ts}}|$ is the number of examples in the test set, and $|m(x;\theta_t) \neq y|$ is one if the model prediction does not match the ground truth label and zero otherwise. The outer sum computes the discrete integral of the error rate over time. CER is going to be small only when the error rate is small throughout the entire stream. CER is instead large for a tardy model that waits till the very last mega-batch to update the model, even though eventually this may obtain a very low final error rate.

Similarly, we compute the cumulative memory and compute usage as:

$$\text{Mem} = \sum_{t=0}^{B} |\theta_t|, \quad \text{Comp} = \sum_{t=0}^{B} \mathcal{O}(m(\cdot;\theta_t)), \tag{7.2}$$

Algorithm 1 Training in the ALMA setting

procedure TRAIN(m, w, replay, grow) ▷ m is the model, w is the waiting time
  $t \leftarrow 1$
  $\mathcal{D} \leftarrow \emptyset$
  while $t < B$ **do** ▷ For each stage
    if replay **then** ▷ Acquire w mega-batches
      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_t \cup ... \cup \mathcal{D}_{t+w-1}$
    else
      $\mathcal{D} \leftarrow \mathcal{D}_t \cup ... \cup \mathcal{D}_{t+w-1}$
    $t \leftarrow t + w$
    if grow **then**
      $m.grow()$ ▷ Grow the model if the model is a growing model
    $m.train(\mathcal{D})$ ▷ Fine-tune or retrain from scratch m on the collected dataset

where $|\theta_t|$ is the number of free parameters of the model at time t, and $\mathcal{O}(m(\cdot; \theta_t))$ is the number of flops used by the model to process the t-th mega-batch. Notice that the above metrics are measured by the environment as training progresses, and will be used in our empirical assessment (§7.3). However, the learner does not have access to the test set. The learner has only access to the validation set of the current mega-batch, and can only use that to select its own hyper-parameters.

7.2 Learning Algorithms

In this section, we describe the methods we tested in the ALMA setting. They generally follow the learning procedure shown in Algorithm 1. At a high level, we consider two families of models, those with a monolithic architecture and those with a modular architecture (e.g. ensembling). The latter are amenable to grow over time by adding new modules to the existing set. We will start by describing fixed architectures (§7.2.1) and then conclude with growing architectures (§7.2.2). We also evaluate models in the setting where they can replay previous mega-batches.

7.2.1 Fixed Architectures

The first family of methods trains models with a fixed architecture. These models are sequentially trained over new mega-batches and exhibit a fixed memory footprint. We consider three models:

Single Model (*SM*): This is a standard multi-layer neural network (e.g., fully connected neural network or transformer) trained by stochastic gradient descent and initialized from the parameters of the model trained on the previous mega-batch, unless otherwise specified.

Ensemble of Models (*Ens*): The second approach is the simplest modular approach, consisting of an ensemble of N neural networks with the same architecture but different random initialization seed, each being trained independently on the same data. The output of the overall model at test time is the average probability distribution produced by each component[2]. The advantage of *Ens* is that training and inference can be trivially parallelized, enabling to scale up model parameters very easily. The disadvantange is that inference requires N times more compute than what is required by each component.

Uniform Mixture of Models (*UMix*): A potential drawback of *Ens* is that evaluation and training are inconsistent, meaning that training and testing use different model predictors. *UMix* addresses this by training a model whose prediction is the average (in logit space) of the predictions produced by N networks. While this requires synchronization during training, now both training and evaluation use the same model.

[2]Classical bagging approaches and majority vote strategies have been also explored without significant difference.

7.2.2 Growing Architectures

In the previous section, the number of parameters and the architecture of the model are fixed throughout the model's lifetime. However, as more data is observed, it is interesting to consider dynamic architectures that grow over time, because these may save compute and memory during the earlier stages of learning while providing more predictive power during the later stages. We consider three growing approaches:

Growing Ensemble (*gEns*): Like the *Ens* model, *gEns* is also a combination of neural networks trained independently. While *Ens* considers a fixed number of networks that are, at each stage, trained over the new chunck of data, *gEns* replaces this step by a growing step where k new neural networks are added. In our implementation, only these k neural networks are trained over the new data, while the other neural networks (trained on previous mega-batches) are kept fixed. Therefore, when starting with a single component and until the next growing step, the cost of training *gEns* is equal to *SM* for the same model architecture. Unless otherwise specified, we use $k = 1$ for the experiments in the chapter.

Growing Mixture of Experts (*gMoE*): A hierarchical mixture of experts models (MoE) is an architecture where at layer l the output representation is $z^l = \sum_{j=1}^{k} g(j|z^{l-1})h(z^{l-1}|j)$, where g is the gating or routing function and $h(\cdot|j)$ is the j-th expert. Compared to *Ens*, MoE has exponentially many more components albeit with a lot of parameter sharing. Another advantage is that when selecting only one (or a few) experts, the computational cost is independent of the number of experts, assuming the cost of gating is negligible compared to the cost or executing the experts. The main issue is that MoE are notoriously harder to train [Eigen et al., 2014, Denoyer and Gallinari, 2015, Lepikhin et al., 2020]. In this work, we consider a growing version of MoE, which we denote with *gMoE*, whereby experts are added gradually over time. This has a tree structured

gating function where leaves correspond to experts. At each layer, we calculate each expert's contribution to the total loss by summing the losses of the examples routed through that expert. We then "split" the expert responsible for the largest contribution to the loss. The split is performed by adding an expert with the same parameters, and turning the corresponding leaf node of the gate into a binary internal node with a child leaf for the old and new expert. This process guarantees that right before and right after a growth step the loss is the same. See the appendix for further details.

Firefly [Wu et al., 2020] (*FF*): *FF* is a method which progressively grows neural networks, jointly optimizing both the model architecture and parameters. Growth includes both a width expansion by adding new hidden units (or feature maps) as well as a depth expansion by adding new layers. Importantly, this is an example of non-modular method unlike *Ens* or *gMoE*, which is potentially more expressive but also more inefficient at inference time because there is no structured sparsity that can be leveraged to speed up computation.

7.3 Experiments

In this section we first describe how standard benchmarks can be repurposed for ALMA, we then provide the details of the models we tested, and we finally conclude with an analysis of the results we obtained, aiming to understand which method attains the best trade-off between time, accuracy, compute and memory usage.

Datasets We consider a variety of datasets. The first dataset is CIFAR 10 [Krizhevsky, 2009] that has a training set with $50{,}000$ images of size 32x32 pixels belonging to 10 classes such as bird, car, horse, ship, truck, etc. The second dataset is MNIST [LeCun et al., 1998], which consists of a training set with $60{,}000$ quasi-binary handwritten digits of size 28x28 pixels, and a test set with $10{,}000$ examples. The third dataset, used for our

large-scale language modeling evaluation, is a portion of the collection of English language text introduced in Liu et al. [2019], consisting of Books, Wikipedia and Common Crawl. We consider 4 (large) mega-batches for training and one additional mega-batch for evaluation, each consisting of approximately 440M words; we also hold out a validation set with approximately 0.5M words of Common Crawl for model selection. We use a byte-pair encoding (BPE) [Sennrich et al., 2016] vocabulary with $50{,}000$ units, following Radford et al. [2019]. This dataset is fairly representative of what practitioners might face when maintaining a deployed system with new data arriving every few months.

Given a dataset like any of the above, we construct a benchmark for ALMA evaluation as follows: 1) we randomly partition the training set into B mega-batches with equal number of training examples ($B = 100$ for CIFAR, $B = 500$ for MNIST and $B = 4$ for the text dataset), 2) from each mega-batch we extract 10% of the data to build the mega-batch validation set (except for the large scale language modeling dataset where we use the provided validation set), and 3) we create a learning experience by doing one pass over the sequence of mega-batches. For each mega-batch, the learner can query as many mini-batches as desired. The learner can also decide not to train on the data of a mega-batch right away but instead *to wait* and accumulate data across a few consecutive mega-batches. While the learner observes data, it is also tested on the test set. This is not used for validation purposes, but only for final reporting as shown in §7.4.

Models We evaluate the six approaches presented in §7.2, and for each of them we consider various waiting times, a version with and without replay, and at least four model sizes. For methods with expanding architectures, we try different configurations of hyper-parameters controlling *when* to grow, and *how much* to grow. For simplicity, we limit expansion phases to occur in between megabatches. Next, we describe

in detail the architecture used on each dataset. Further experimental details to aide reproducibility are reported in the appendix. On MNIST the backbone architecture of *SM* is a three layer fully connected neural network with ReLU units. We considered various hidden units size, ranging from 4 to 64 (which we refer to as [small] and [large], respectively), which let us simulate the regime of big data relative to the size of the network and explore how to grow architectures without worrying about overfitting. Similarly, the components of *Ens*, *gEns* and *UMix* are *SM* networks of the same size as stated above; *gMoE* also starts off as *SM* and adds modules (at the first two layers) that have the same size as the original layer of *SM*.

On CIFAR 10, the methods and notations are the same as in MNIST. The only difference is that the backbone architecture is a scaled down version of a VGG19 convolutional neural network [Simonyan and Zisserman, 2015] as in [Wu et al., 2020], where the number of intermediate feature maps is the same for each layer, ranging from 4 to 64. On this dataset, we also consider *FF* starting off from the same VGG19 backbone.

For the language modeling task *SM* is a Switch Transformer [Fedus et al., 2021], which is a hard mixture of experts model with an additional load balancing loss term and hard capacity constraint applied during training to prevent uneven expert utilization. Following Fedus et al. [2021], we fix the weight of the balancing loss term to 0.01 and use a capacity factor of 1, ensuring relatively uniform expert utilization. We train the model using Adam [Kingma and Ba, 2015] and tune the learning rate and dropout on the validation set. In the growing setting we copy the expert weights and gating network weights corresponding to the top-k experts incurring the largest loss, where k is typically between 2 and 4. This growing procedure preserves a flat mixture and adds multiple experts at the time. While this approach performs slightly worse than the one described in §7.2.2, it is easier to implement at scale. We consider two model sizes: a *base* model with 6 layers and model dimension of 512, for a total of 40M shared parameters and 6M additional parameters per expert; and a *large* model with

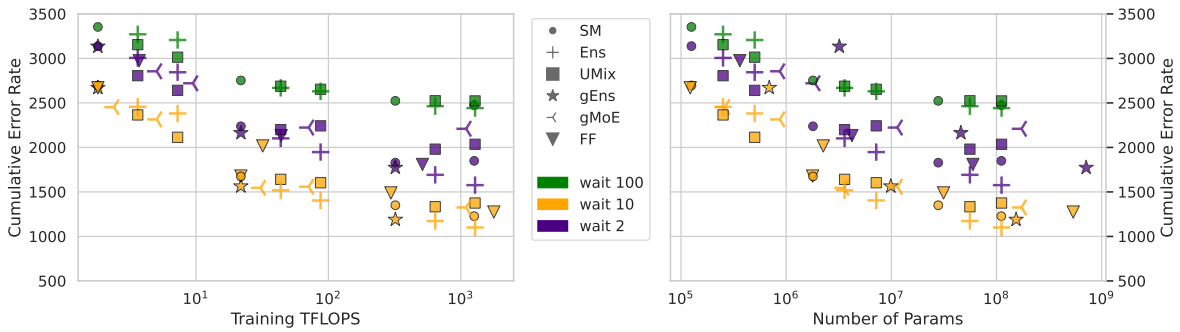

Figure 7.2 CIFAR 10 results: Cumulative error rate versus cumulative flops and number of parameters without replay. For the same model type we vary the size of the backbone architecture and the waiting time.

12 layers and model dimension of 768, for a total of 96M shared parameters and 28M additional parameters per expert. We use an input sequence length of 512 tokens and we do not use replay given the large mega-batch sizes. During each mega-batch, we train all language models for exactly 120000 gradient steps (results in Fig. 7.5) unless otherwise specified (e.g. Tab. 7.1). This makes it easier to compare models for the same computational budget at the mega-batch level.

7.4 Results

7.4.1 Visual Recognition

Since conclusions are rather similar, we focus our analysis on the more challenging CIFAR 10 dataset, and report results also on MNIST in the appendix.

Smallest waiting time might not be optimal: We begin our analysis in the setting without replay, shown in Fig. 7.2. We first observe that an intermediate waiting time (in this case equal to 10) strikes the best trade-off between Cumulative Error Rate (CER) and both training cost (left) and memory cost (right). As shown in Fig. 7.3-top, where the test error rate is plotted as a function of the number of mega-batches received, greedy methods using waiting time equal to 2 achieve a lower error rate only

during the very beginning of the stream, but are outperformed later on. Tardy predictors waiting for all 100 mega-batches before training obtain the best final accuracy, but have no predictive capabilities throughout the first 99 mega-batches. Instead, methods with an intermediate waiting time (shown in orange) can quickly deliver a reasonable predictor early in the stream, and obtain a final error rate that is very close to the lower bound obtained by tardy methods. Thus, a waiting time of 10 yields the lowest area under the curve (or CER) on CIFAR 10.

On MNIST however, an intermediate waiting time is best only for small models, as shown in Fig. 7.3-bottom. Very greedy models do not converge as well in this setting, which leads to a significant penalty in terms of CER. However, bigger networks converge very fast in just a few megabatches, making smaller waiting times more desirable. To summarize, we can break down the relative loss encountered by different waiting times in two phases, *before* and *after* training on the first mega-batch. For easy settings (e.g. MNIST with a large model), most of the error is accumulated before the first training phase, therefore smaller wait times are preferred. For harder settings (e.g. CIFAR), because models trained prematurely underperform in the limit, given a long enough stream, a small wait time will be suboptimal. Therefore, the optimal waiting time depends on several factors, namely the time horizon and how quickly the model learns, which is itself a function of design choices such as the model size. We expect these findings to generalize to other tasks (beyond supervised learning) and modalities (beyond images) in the ALMA setting. In any case, in such non-convex setting, it is certainly not true that learning on the data as soon as it becomes available always attains the best trade-off between error rate and compute.

Larger models are more statistically efficient: It was shown in Kaplan et al. [2020] that for transformer models trained on natural language, bigger models need fewer samples to reach a given performance. We confirm that a similar trend is also observed in the ALMA setting, and that statistical efficiency of larger models is also true for

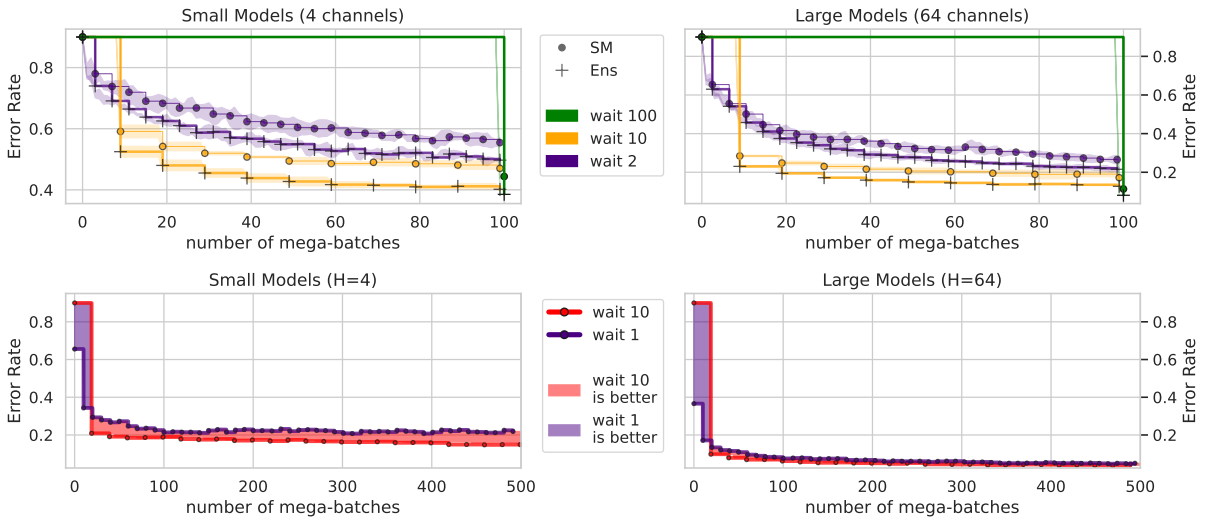

Figure 7.3 Error rate over time of small models (left) and large models (right) on CIFAR 10 (top) and MNIST (bottom).

other modalities (images) and architectures (CNNs, MLPs). Indeed, we observe that bigger models, *SM* and *Ens*, not only generalize better but they are also statistically more efficient: on the small *Ens* obtained almost 40% error rate by the end of its learning experience (Fig. 7.3-top left), which is worse than the error rate obtained by the large *Ens* just after having observed one tenth of the entire stream. We obtained similar results on MNIST (see Fig. 7.3-bottom) and convolutional models, showing that MLPs and CNNs also benefit from the same statistical efficiency gains when scaling the parameter count. Overall, we expect that this finding, stating that the statistical efficiency of large models applies beyond NLP and transformers, to generalize to other tasks and domains in the ALMA setting.

Growing does not improve: We first investigate the efficiency of growing, since in principle, we would expect that adapting model capacity with the amount of data should strike a better trade-off between accuracy and memory/compute usage. Growing has been explored in Continual Learning as a means to mitigate forgetting; by adding new parameters and keeping old ones fixed, there would be no knowledge erasure (see sec 3.3.3). The question of whether growing over time is a better allocation

of a computation budget has not been studied previously, to the best or our knowledge. We find that for a fixed computation or memory budget, it is always better to **start with a larger model**, rather than growing it over time. Indeed, we find that on both graphs of Fig. 7.2, *SM* almost always outperforms *gMoE* and *FF*, a trend that is especially visible for higher budgets of TFLOPS and parameters. In other words, a *gMoE* or *FF* that starts small and finishes big will typically be outperformed by a *SM* model of average size.

Next, if we focus our attention on the three approaches with growing architectures, namely *gMoE*, *gEns*, and *FF*, we find that there is no clear winner among them. When comparing across a fixed computational budget (Fig. 7.2 left), *gEns* overall performs better than *gMoE* and *FF*. However, when we fix the memory budget instead (Fig. 7.2 right), *gEns* is, on average the worst performing method. Finally, *Ens* is more efficient than *gEns* in terms of memory, but vice versa in terms of training compute. However, should we look at the inference cost of both methods, we would find that *Ens* outperforms its growing counterpart, whose inference cost grows over time while it is fixed for *Ens*. Once again, the best strategy is to pick the largest fixed-capacity model for a given computational budget. Notice that these conclusions apply to the methods considered in this study, and improving approaches that dynamically adapt their architecture over time is clearly a worthwhile avenue of future research. Whether this finding generalizes to sequential learning settings with distribution shifts is beyond the scope of this paper.

***Ens* strikes the best trade-off:** More generally, ***Ens* is the best performing method for larger models** across all our experiments, including the language models reported in §7.4.2. This is a remarkable finding given the simplicity of the approach and how easy it is to parallelize their training process. Ensembling makes it very easy to increase model capacity early on, and it is so far the best way to utilize compute at scale, a possible indication of the inefficiency of training large models using alternative approaches,

which highlights yet another worthwhile avenue of future research.

Replaying past mega-batches does not improve: We now consider the same approaches as before but with models trained on all megabatches seen so far. Therefore, at the very last training step, models are trained on the entire dataset (concatenation of all megabatches). In Fig. 7.4 we report the results when the waiting time is equal to 10. In all cases, replaying data gives better results at the expense of an increase in compute. Except for *gEns*, these gains are roughly the same for all methods, as all segments are parallel to each other. *gEns* gains less as the last component which is trained on the full dataset has disproportionate influence in the model average which includes components trained on fewer megabatches. However, this last component essentially coincides with *SM* trained on the full dataset. Hence the two methods converge to the same performance when using replay. We provide additional results with replay in the appendix, which shows that there are benefits from replaying only at higher computational budgets where also the optimal waiting time reduces to 1.

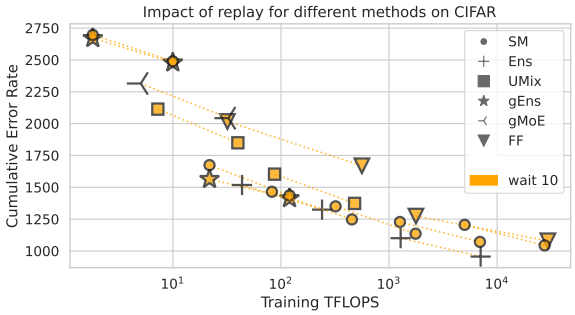

Figure 7.4 Impact of replay on the CIFAR-10 dataset with a wait time of 10. For each method we show a line from the result without replay (left) and with replay (right).

More importantly, we observe that **replay does not yield a significantly better trade-off** between CER and compute. For the same computational budget, methods using replay attain similar CER of methods that do not use replay. Other factors such as the size of the backbone architecture or the waiting time matter more.

7.4.2 Language Modeling Experiments

For the large-scale language modeling experiments, we consider two model sizes (base and large, see §7.3), with an inference cost per input of 42 and 126 GFLOPS, respectively. The number of experts is set to 4, 8 and 12 for *SM*, and it does not affect the inference cost since only one expert per input is selected regardless of the total number of experts. Due to the computational burden of these experiments (in total more than 200 GPU days), we limit our analysis to four mega-batches. Nevertheless, this scale (of model and data) and type of application are rather representative of a typical ALMA setting.

The main results are presented in Fig. 7.5. Each line is a trajectory with four points, one for each mega-batch in the stream, as we report average as opposed to cumulative perplexity. For a given model size and for a given computational budget, there are three *SM* models, one for each number of experts we consider, namely 4, 8 and 12.

Larger models are more efficient: In agreement with our results on computer vision tasks, we observe that bigger models tend to generalize better and are more sample efficient. For instance, the large model after a single mega-batch outperforms all base models, including base models after four mega-batches which have seen four times more data. This finding is consistent across all methods tried for this experiment, and with prior scaling law results [Kaplan et al., 2020].

Growing does not improve: Once again, there is no clear winner among growing methods. For larger models, *gEns* outperforms *gMoE* for the same compute, and perform similarly for base models. However, for all model sizes, *gMoE* is more memory efficient, therefore the optimal approach among them will depend on both compute and memory budget. More importantly, we observe that models with fixed capacity are more compute and memory efficient than models that grow over time. Looking at the average perplexity as a function of the number of experts, we see that methods

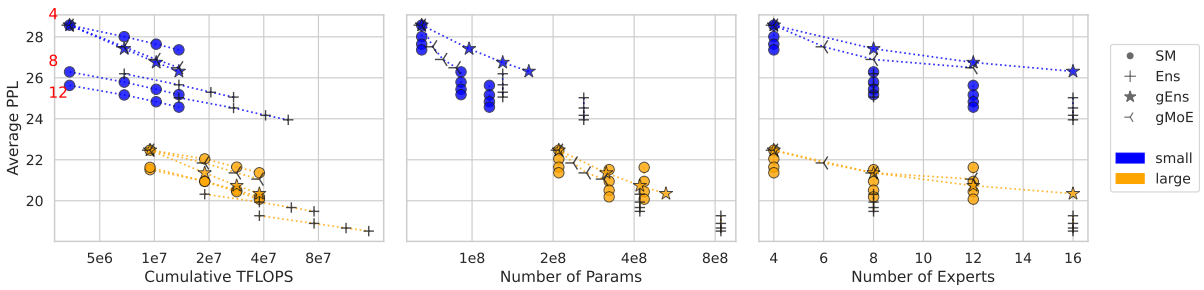

Figure 7.5 Language modeling trade-offs: average perplexity (PPL) versus cumulative compute, number of parameters and number of experts. Numbers in red refer to the number of experts in the corresponding *SM* runs.

which start with a small number of experts and later grow are outperformed by similar fixed architectures which have an intermediate number of experts. This highlights the importance of having more capacity at the start of training, rather than at the end.

Ensembles perform the best: Third, *Ens* thrives in the larger capacity setting. Looking at the orange markers in the graph, we see that for equal computation budget, *Ens* methods outperform all other methods, which is consistent with the computer vision results. In the base setting instead, versions of *SM* (see the lowest blue points) strike a better tradeoff in both compute and memory.

Learning sequentially is harder: We argued initially that once the learner makes several passes over each megabatch, the data distribution cannot be considered i.i.d. anymore, relative to the empirical distribution of the union of all megabatches. It is however unclear how much this issue has a practical impact in the performance of the model. In order to assess this we run one last experiment using our best performing approach, namely *Ens*. We compare a model trained on k mega-batches sequentially with the same model trained all at once on the aggregation of the same k mega-batches. Since both approaches have the same computation budget, the same architecture and are fed with the same data, we can disentangle the effect of the non-i.i.d nature of the data in ALMA. The results shown in Tab. 7.1 confirm that ALMA's sequential (seq.) training is indeed more challenging. Across all four configurations, models incur a

Method	PPL $k=3$, iid	PPL $k=3$, seq.	PPL $k=4$, iid	PPL $k=4$, seq.
Small *Ens* 4@2	**24.30**	24.57	**24.13**	24.35
Big *Ens* 4@2	**18.04**	19.14	**17.88**	18.92

Table 7.1 Ablation on the effect of learning sequentially (seq.) as opposed to learning with fully i.i.d. data, for the same amount of data and compute. The model is an ensemble with 2 components each of which with 4 experts per block.

drop in performance when compared to regular i.i.d training, and even more so when the model is larger. This gap offers another opportunity of future research on ways to make sequential training more effective when using deep non-linear neural networks.

7.5 Related Work

ALMA relates to several other learning frameworks as illustrated on the left of Figure 7.1. i) It shares the same assumptions of classical batch supervised learning [Vapnik, 1998] at the level of each mega-batch. However, it overall violates the assumptions of i.i.d. observations, because data points come in a stream of mega-batches and because the learner typically makes several passes over each mega-batch. Moreover, in ALMA the learner can choose how long to wait before training. In this sense, batch supervised learning can be thought of as an extreme case of ALMA (single mega-batch because learner waited till the end of the stream to train). ii) As mentioned in the previous section, ALMA relates to online learning [Bottou, 1998] because data comes sequentially and because in both cases we measure performance in terms of regret (although in §7.1.1 our cumulative error lacks a reference oracle since this is not known in our setting). However, in ALMA we are also explicit about the computational budget used by the model and aim at striking a good trade-off between regret and computational cost. In our current work, we restrict ALMA to stationary distributions, while online learning is more general and encompasses also non-stationary distributions. Fi-

nally and most importantly, in ALMA we focus on non-linear predictors while typical literature on online learning considers linear predictors. iii) Similarly, ALMA relates to concept drift [Lu et al., 2018] because of the sequential nature of the observations. However, literature on concept drift often focuses on linear predictors. iv) ALMA can be seen as a degenerate case of supervised continual learning, where the task distribution is stationary. However, in supervised continual learning there is often a focus on attaining a predictor that represents the entire distribution of tasks by the end of learning, while in ALMA we measure cumulative error like in prequential learning. v) ALMA relates more broadly to transfer learning [Pan and Yang, 2010], as the problem of adapting to a new batch of data can be interpreted as leveraging knowledge acquired on previous batches to more effciently learn from the new batch of data. vi) Finally, ALMA relates to anytime learning [Grefenstette and Ramsey, 1992, Ramsey and Grefenstette, 1994], which has been recently applied to compare various autoML frameworks [Liu et al., 2020]. However, unlike traditional anytime learning, in this work we are not interested in assessing the anytime learning ability at the level of each mega-batch, but only at a coarser granularity, at the level of the entire stream of mega-batches. Lastly, we note that while anytime learning operates in a similar setting as online learning (see Fig. 7.1), it is often used with non-linear predictors in a supervised learning setting.

To the best of our knowledge, the most relevant prior work is by Sahoo et al. [2018] which considers a setting similar to ours, except that their stream is composed by individual examples and in their setting there is no concept of waiting time nor revisiting data points several times. However, they also benchmark against methods that increase capacity over time, although their analysis was limited to fully connected networks.

7.6 Discussion

In the first part of the chapter, we promised the reader to provide an empirical answer to several questions:

1) *How long should the learner wait before training on the newly arrived mega-batches?* There is no single answer to this question. We have seen that on CIFAR 10 but also on MNIST when using smaller architectures and when using replay with smaller compute budgets, an intermediate waiting time strike the best trade-off. However, there is no known formula for deriving the waiting time, as it depends on several factors such as the time horizon, the initial performance of the model and how quickly a model learns, to name a few. The firm conclusion is that greedily updating the model as soon as data becomes available, as advocated by literature on convex online learning, might not always be the best strategy when using deep neural networks, In practice, also waiting too long, to the point that the learner does not even have time to perform a single pass over the aggregated mega-batches, might be suboptimal.

2) *What architecture should the learner adopt?* Our study indicates that, among all methods we tested, ensembling strikes the best trade-off in general. Ensembling is simple and easily parallelizable, and it offers a straightforward way to increase capacity. Starting off with a larger model, for instance via ensembling, is an excellent way to obtain good anytime performance.

3) *Should the learner increase capacity over time as more data is observed?* The answer is negative, currently. It is better to start off with the largest architecture fitting into memory and keeping that fixed. A cynical interpretation of this conclusion could make the reader believe that growing the architecture size should not be a topic of interest. However, as data is added over time so is computation and memory. It is often the case that researchers working on large-scale learning instantiate (rightly so) the biggest possible model to train on their task, but few months later they can manage to launch

even bigger models thanks to compute and engineering advances. How can the larger model leverage what has been learned from the previously trained model? Is there a modeling choice that strikes a better trade-off than retraining from scratch? More generally, what are good approaches to extract information from a new batch of data to integrate it into an existing model? We believe these are great avenues of future research, and that our ALMA framework (learning and evaluation protocol, codebase, baselines) provides a good abstraction of the practical setting, and a sound tool to pursue such investigation.

In conclusion, this chapter takes a step towards solving a more realistic continual learning problem and its corresponding constraints. Nevertheless, I would like to point out several limitations of the analysis presented in this chapter. Firstly, as we have seen in the experiments (e.g. Fig. 7.3), the first training run, starting from a randomly initialized model, has a disproportionate impact on the cumulative error rate. Indeed, because the model outputs random predictions before initial training, there is a strong incentive to use a smaller waiting time. Given that across many modalities there exists foundation models which can serve as a good initialization, leveraging such pretrained weights could enable practitioners to use longer wait times. Second, the analysis in the chapter focused on settings where mega-batches are drawn from the same underlying distribution. How would our findings change should if we were to introduce a distribution shift over the mega-batches ? We know that this chapter's finding stating that replay is not generally helpful in ALMA does not hold in non-stationary settings. We therefore caution the reader in extrapolating the presented results to other learning settings.

Chapter 8

Multi-Head Adapter Routing for Cross-Task Generalization

In the first chapter of this thesis, we proposed a two pronged solution to build efficient continual learners. In chapters 4, 5, 6, we focused on improving the first component, *backward compatible learning*, by designing more efficient replay-based methods. In the previous chapter, we shifted our attention towards the second component, *forward transfer*, and performed an extensive empirical evaluation of several strategies under sequential learning benchmarks. However, in most of the results presented so far, the learning settings and solutions were still somewhat detached from realistic use cases. Indeed, the benchmarks, while interesting, were artificially adapted to sequential settings, and the starting models were usually initialized randomly. In this penultimate chapter, we continue towards enabling better forward transfer, and tackle a challenging realization of continual learning which does not suffer from the prior drawbacks: *cross-task generalization*. This setup works as follows : we are given a pretrained model, and additional multi-task training data, and the goal is to leverage both model and data for *few-shot adaptation to new tasks*. Indeed, the ability to train effective models with a

relatively small number of training data is of paramount importance due to the paucity of annotated examples for most tasks. One effective few-shot learning approach is to leverage large models pre-trained on a vast amount of unlabelled data and fine-tune them on the few examples available for each downstream task. Consider the following example of cross-task generalization. You have access to a generalist model, such as GPT-4, and data collected from several thousands of users (tasks) interacting via text on social media, or sequential gaming data from online gaming platforms. How can we build a system which is not only efficient, but can also quickly adapt to new users for whom very little data is collected ?

To reduce the memory cost of duplicating the entire array of parameters for each downstream task, recent approaches resort to parameter-efficient fine-tuning (PEFT) methods, such as LoRA [Hu et al., 2022], SFT [Ansell et al., 2022], or $(IA)^3$ [Liu et al., 2022]. These only fine-tune adapters while leaving the pre-trained model "frozen". From a continual learning perspective, because new parameters are trained and old ones frozen, there is no risk of catastrophic forgetting. This is only achievable because at inference time, we know which PEFT adapters to retrieve for a given task or user.

Nevertheless, it remains unclear how to best exploit a set of *training* tasks to better generalize to a set of unseen *test* tasks in a sample-efficient fashion, based on just a few examples. One straightforward solution is to perform multi-task pre-training, i.e. first train the large model on the union of the examples from the training tasks, then fine-tune the obtained model to the test task [Liu et al., 2022, Ye et al., 2021]. However, this solution does not take into account that test tasks may require solving different combinations of sub-problems compared to training tasks [Vu et al., 2020], thus failing to achieve compositional generalization [Rosenbaum et al., 2019, Ponti, 2021]. Moreover, specializing the model towards different tasks during training may result in negative transfer, due to their corresponding gradients being misaligned [Wang et al., 2021].

Several PEFT approaches have been proposed to enable better cross-task gener-

alization by training adapters (or soft prompts) on each task independently [Pfeiffer et al., 2021, Vu et al., 2021, Asai et al., 2022, Chronopoulou et al., 2023]. Given a new test task, parameters from similar training tasks are aggregated, which enables transfer. While solely having task-specific parameters is an effective strategy to mitigate interference across training tasks, it also inhibits any positive transfer within the same task pool. Polytropon (`Poly`) was recently proposed by Ponti et al. [2023] to address these issues: the model assumes that task-specific adapters are learned combinations of a reusable inventory of basis adapters or *modules*. In practice, each module is implemented as a LoRA [Hu et al., 2022] adapter, which modifies a large pre-trained model, such as T5 [Raffel et al., 2020]. During both multi-task pre-training and few-shot adaptation, `Poly` learns both the inventory of adapters and a (continuously relaxed) binary task–module routing matrix, which determines which module is active for each task. While `Poly` shows promising results, several questions remain unanswered: 1) Does the expressivity of the routing function matter? 2) Why do routing-based PEFT methods yield superior performance? 3) Is routing useful during both multi-task pre-training and few-shot adaptation?

To answer the first question, we propose a new routing function, `MHR`, that mixes adapters at a more granular level. Differently from `Poly`, where routing decisions are made for each adapter as a whole, in `MHR` we linearly combine subsets of the adapter dimensions (i.e. heads), each with different combination coefficients. We evaluate `MHR` and a series of competitive baselines for few-shot task adaptation on the T0 task suite [Sanh et al., 2022] and Super-Natural Instructions [SuperNI; Wang et al., 2022c]. Based on our results, we report that `MHR` outperforms `Poly` and single adapter baselines. Additionally, we show that, thanks to the increased expressivity of the routing function, it becomes possible to fine-tune only the parameters of the routing function (and not the adapters) during few-shot adaptation: the resulting method, `MHR-`z, yields competitive performance while requiring orders of magnitude fewer parameters.

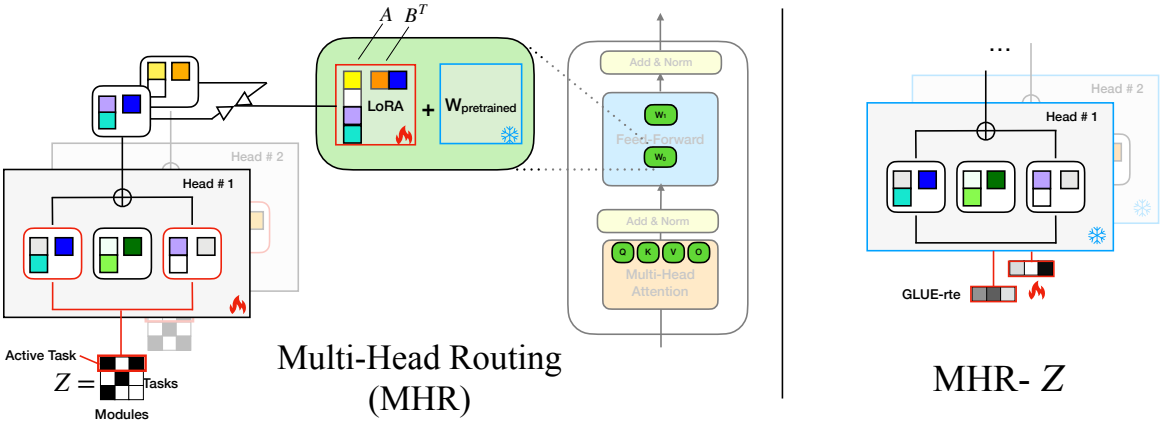

Figure 8.1 *Left:* A LoRA adapter with weight $AB^\top$ is trained on top of a frozen, pre-trained linear layer W. Our method `MHR` partitions the A, B parameter indexes into h subsets (or *heads*). For each subset, a separate routing function selects the active modules for the current task among m copies with different parameter values, and combines them via averaging to form a task-specific head. The heads are then concatenated to form the LoRA adapter. Using multiple heads allows for more fine-grained mixing of task parameters with a negligible increase in overall parameter count. *Right:* During few-shot adaptation, one can fine-tune only the multi-head routing parameters (`MHR-`z), keeping the modules frozen, resulting in highly parameter-efficient adaptation.

Regarding the second and third questions, we uncover that optimization during multitask pretraining plays a key role in explaining the downstream performance of routing-based PEFT approaches. Specifically, we find that `MHR` exhibits a higher cosine similarity between gradients from different tasks than `Poly` and single-adapter multi-task training. Hence, routing enables more knowledge transfer and less interference across tasks during multi-task pre-training. This finding led us to investigate whether routing is useful also during few-shot adaptation. It has been hypothesized [Ponti et al., 2023] that one of the reasons behind `Poly`'s performance resides in the inductive bias of the modular architecture, which allows test tasks to recombine and locally adapt the most relevant modules. To test this hypothesis, we propose `MHR-`μ, where the routing function is discarded and all available adapter parameters are averaged before few-shot adaptation. We find that `MHR-`μ can recover the performance of

MHR, hinting that Poly/MHR gains are only a result of better multi-task optimization. Finally, we show that MHR-μ can also be used as an effective zero-shot transfer method by training the average of the pre-trained adapters for a few additional steps on the multi-task training set. This yields gains up to 3% on absolute accuracy w.r.t. to strong baselines such as T0-11B.

8.1 Technical Background

8.1.1 Transformer Models

Introduced by Vaswani et al. [2017], Transformers are a type of neural network architecture that has been highly successful in natural language processing tasks. One of the key components in the Transformer architecture is the self-attention mechanism, which allows the model to weigh and aggregate information from different parts of the input sequence. This removes the need for recurrent connections, making training fully parallelizeable across the sequence.

Self-Attention The self-attention mechanism can be described as follows: Given a sequence of input vectors $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$, the self-attention mechanism computes a weighted sum of these vectors for each position in the sequence. The weights for each position are determined by the compatibility between the input vectors, which is calculated using the dot product of the query, key, and value vectors. First, the input vectors are linearly transformed into query $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V \tag{8.1}$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are learnable weight matrices. Next, the compatibility scores between each pair of positions in the sequence are calculated using the dot product of

the corresponding query and key vectors, followed by scaling with a factor $\sqrt{d_k}$, where d_k is the dimension of the key vectors:

$$\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}. \tag{8.2}$$

In the case of autoregressive modelling, one can enforce temporal consistency by ensuring that $\mathbf{S}_{ij} = -\infty \; \forall j > i$, i.e. disabling tokens from attending future positions in the sequence. The compatibility scores are then transformed into a probability distribution using the softmax function: $\mathbf{A} = \text{softmax}(\mathbf{S})$.

Transformer Block Finally, the self-attention output is computed as the weighted sum of the value vectors, with the weights given by the attention probabilities: $\mathbf{Y} = \mathbf{AV}$. In the multi-head attention mechanism, the input vectors are split into multiple heads, each of which applies the self-attention mechanism independently. The resulting output vectors are then concatenated and linearly transformed to obtain the final multi-head attention output.

In order to build a full Transformer Model, the attention layers can be stacked on top of each other to emulate a multi-layer neural network. Typically, a feed-forward layer, layer normalization [Ba et al., 2016] and residual connections are used after one attention layer, referred to as a *transformer block*, as depicted in Fig. 8.2.

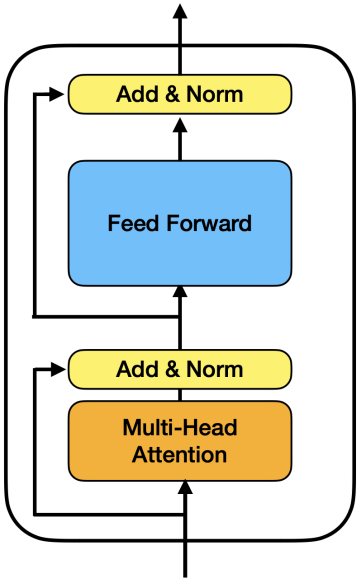

Figure 8.2 A Transformer Block. "Add and Norm" represent a skip connection followed by a layer normalization operation.

8.1.2 Adapters: LoRA & (IA)^3

LoRA [Hu et al., 2022] and (IA)^3 [Liu et al., 2022] are two recently proposed adapter architectures that achieve competitive trade-offs between performance and parameter efficiency [Karimi Mahabadi et al., 2021, Liu et al., 2022]. For each linear transformation corresponding to the query (q), key (k), value (v) and output (o) of the self-attention layers, LoRA modifies the base model *parameters* as follows:

$$\boldsymbol{h}^{q,k,v,o} = \boldsymbol{W}_0^{q,k,v,o} x + s \cdot \boldsymbol{A}^{q,k,v,o} (\boldsymbol{B}^{q,k,v,o})^\top x, \tag{LoRA}$$

where $\boldsymbol{W}_0$ are the (frozen) weights of the pre-trained model (e.g. T5 [Raffel et al., 2020]). $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d \times r}$ are low-rank learnable parameters and $s \geq 1$ is a tunable scalar hyperparameter. (IA)^3, on the other hand, modifies key and value *representations* in self-attention element-wise, and also modifies the feed-forward MLP (f):

$$\boldsymbol{h}^{k,v} = \boldsymbol{l}^{k,v} \odot (\boldsymbol{W}_0^{k,v} x); \quad \boldsymbol{h}^{f} = (\boldsymbol{l}^{f} \odot \gamma(\boldsymbol{W}_1^{f} x)) \boldsymbol{W}_2^{f}, \tag{(IA)^3}$$

where $\boldsymbol{l}^{k,v,f} \in \mathbb{R}^d$ are learnable parameters , $\boldsymbol{W}_{1,2}^{f}$ the frozen parameters of the feed-forward layer in the backbone, and γ a non-linearity. For clarity, we will drop the superscripts q, k, v, o in the rest of the chapter.

8.1.3 Polytropon: Adapter Routing

Typical adapter methods either fully share adapters across tasks or train individual adapters for each task. `Poly` addresses the multi-task problem by softly sharing adapter parameters across tasks. Each `Poly` layer contains 1) an inventory of adapter modules $\mathcal{M} = \{\phi_1, \ldots, \phi_m\}$ with $|\mathcal{M}| \ll |\mathcal{T}|$; 2) a routing function $r(\cdot)$ that chooses which subset of the modules to combine for each task.

Each module corresponds to a LoRA adapter, where ϕ_i are its associated parame-

ters $\boldsymbol{A}^{(i)}, \boldsymbol{B}^{(i)} \in \mathbb{R}^{d\times r}$. $r(\cdot)$ is implemented as a task–module routing matrix $\boldsymbol{Z} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{M}|}$. $z_\tau = \boldsymbol{Z}_{\tau,:} \in \mathbb{R}^{|\mathcal{M}|}$ is a routing vector of task $\mathcal{T}_\tau$, with cell $Z_{\tau,j}$ being the probability logits of using module ϕ_j for task $\mathcal{T}_\tau$ in the current layer. Differently from mixture-of-experts [Fedus et al., 2021], which perform token-level top-k routing, $\boldsymbol{Z}$ converges to a binary matrix, defining a soft partition over modules. This is achieved by using a Gumbel-sigmoid distribution [Jang et al., 2017] during training, with $\hat{Z}_{\tau,j} \sim \texttt{Gumbel}(Z_{\tau,j})$. At each forward pass, `Poly` can be defined as :

$$\boldsymbol{A}^\tau = \sum_i \alpha_i \boldsymbol{A}^{(i)}; \quad \boldsymbol{B}^\tau = \sum_i \alpha_i \boldsymbol{B}^{(i)}, \tag{Poly}$$

where $\alpha_i = \frac{\hat{Z}_{\tau,i}}{\sum_j \hat{Z}_{\tau,j}}$, and $\boldsymbol{A}^{(i)}, \boldsymbol{B}^{(i)}, \boldsymbol{A}^\tau, \boldsymbol{B}^\tau \in \mathbb{R}^{d\times r}$. We normalize the mixing coefficients $\hat{Z}_{\tau,i}$ for each task to ensure that the number of active modules does not affect the norm of $\boldsymbol{A}^\tau, \boldsymbol{B}^\tau$. Overall, this approach enables different subsets of *modules* to be activated for the current layer and combined in a task-specific way. Following LoRA, the output of the `Poly` layer is added to the output of the original layer of the frozen backbone: $\boldsymbol{h} = \boldsymbol{W}_0 x + s\boldsymbol{A}^\tau(\boldsymbol{B}^\tau)^\top x$.

During multi-task pre-training, for each query, key, value, and output projection in self-attention layers, the parameters learned by `Poly` are the adapter parameters, $\{\boldsymbol{A}_i, \boldsymbol{B}_i\}_{i=1}^{|\mathcal{M}|}$, and the routing matrices $\boldsymbol{Z}$. During fine-tuning, for each test task τ, `Poly` randomly initializes the routing vector $z_\tau \in \mathbb{R}^{1\times|\mathcal{M}|}$ and fine-tunes both z_τ and all the modules parameters $\mathcal{M}$.

8.2 Learning Setting

In cross-task generalization, we are given a set of tasks $\mathcal{T} = \{\mathcal{T}_1, .., \mathcal{T}_{|\mathcal{T}|}\}$, with each task $\mathcal{T}_i$ dataset containing a set of samples $\mathcal{D}_i = \{(x_1, y_1), ..., (x_n, y_n)\}$. The set of all tasks is partitioned into training and test tasks, $\mathcal{T} = \mathcal{T}_{train} \cup \mathcal{T}_{eval}$, and the objective is to lever-

Method	Pre-Training	Fine-Tuning	Inference
`Full FT`	$d \times d$	$d \times d$	$d \times d$
`LoRA`	$d \times 2r$	$d \times 2r$	$d \times 2r$
`Poly`	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{T}\rvert \times \lvert\mathcal{M}\rvert$	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{M}\rvert$	$d \times 2r$
`Poly-`z	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{T}\rvert \times \lvert\mathcal{M}\rvert$	$\lvert\mathcal{M}\rvert$	$\lvert\mathcal{M}\rvert$
`MHR-`μ	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{T}\rvert \times \lvert\mathcal{M}\rvert$	$d \times 2r$	$d \times 2r$
`MHR-`z	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{T}\rvert \times \lvert\mathcal{M}\rvert \times h$	$\lvert\mathcal{M}\rvert \times h$	$\lvert\mathcal{M}\rvert \times h$
`MHR`	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{T}\rvert \times \lvert\mathcal{M}\rvert \times h$	$d \times 2r \times \lvert\mathcal{M}\rvert + \lvert\mathcal{M}\rvert \times h$	$d \times 2r$

Table 8.1 Number of parameters (per layer) used for each method. The calculation uses `LoRA` as the base adapter, modifying a linear transform in $\mathbb{R}^{d \times d}$. Note that the total number of parameters changed by `Full FT` is larger, given that the method also changes parameters for layers not modified by `LoRA`.

age data in $\mathcal{T}_{train}$ and transfer knowledge to facilitate learning of the test tasks $\mathcal{T}_{eval}$. For all the methods discussed, learning takes place in two phases, excluding the original unsupervised pre-training of the language model backbone on a separate corpus. The first phase consists of multi-task pre-training, in which either an adapter, such as LoRA or $(IA)^3$, or the full backbone is trained on the set of training tasks $\mathcal{T}_{train}$. The second phase consists in few-shot adaptation, where the learned adapters are fine-tuned independently on each test task in $\mathcal{T}_{eval}$. We follow the procedure from [Raffel et al., 2020] and formulate each task as a text-to-text problem, enabling standard maximum-likelihood training with teacher forcing [Bengio et al., 2015] and a cross-entropy loss.

8.3 Multi-Head Adapter Routing (`MHR`)

In `Poly`, module combination remains *coarse*: only linear combinations of modules are possible, and thus the resulting aggregated adapter remains a linear function of the modules. We propose to augment the expressivity of the module combination while keeping the parameter count similar. `MHR` (Fig. 8.1) takes inspiration from multi-head attention [Vaswani et al., 2017]: it partitions the input dimensions into h different dis-

joint subsets, performs a separate Poly-style combination for each of them, and finally concatenates them. This corresponds to learning a different routing matrix $\boldsymbol{Z}$ for each subset of input features, therefore enabling the model to select different adapters for different subsets of the input dimensions. This aggregation approach is *piecewise* linear (i.e., linear within disjoint intervals), which allows for more expressive combinations of modules.

In each MHR layer, the routing function is a third-order tensor $\mathbf{Z} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{M}|\times h}$, where $\mathbf{Z}_{:,:,h} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{M}|}$ is a 2D slice of the tensor $\mathbf{Z}$. A slice represents the routing matrix for each of the h heads. Let us denote with $\boldsymbol{W}[k] \in \mathbb{R}^{\frac{d}{h}\times r}$ the k-th partition along the rows of the matrix $\boldsymbol{W} \in \mathbb{R}^{d\times r}$. The adapter parameters $\boldsymbol{A}^\tau \in \mathbb{R}^{d\times r}$ for task τ, and for each adapter layer, are computed as (similarly for $\boldsymbol{B}^\tau$):

$$\boldsymbol{A}^\tau_k = \sum_j \boldsymbol{A}_j[k] \cdot \frac{\hat{\boldsymbol{Z}}_{\tau,j,k}}{\sum_j \hat{\boldsymbol{Z}}_{\tau,j,k}} \text{ with } \boldsymbol{A}^\tau_k \in \mathbb{R}^{\frac{d}{h}\times r}, \tag{MHR}$$

$$\boldsymbol{A}^\tau = \texttt{concat}(\boldsymbol{A}^\tau_1, \ldots, \boldsymbol{A}^\tau_h)$$

where concat concatenates along the first dimension. Multi-task pre-training and fine-tuning are similar to Poly. Note that MHR results in only a negligible increase in the total amount of parameters, since most of the parameters are contained in the LoRA weights $\boldsymbol{A}, \boldsymbol{B}$ (Tab. 8.1).

Routing-Only Fine-Tuning (MHR-z) Prior work [Shao et al., 2023, *inter alia*] has shown that compositional generalization can be achieved by learning to (re-)combine in novel ways pre-existing modules. We investigate whether fine-tuning the module parameters is really needed for few-shot adaptation in the context of both Poly and MHR. Therefore, we name Poly-z and MHR-z the variants that, during few-shot adaptation, keep the parameters of the modules learned during multi-task pre-training fixed and just update the routing parameters $\boldsymbol{Z}$. Crucially, this enables highly parameter-efficient

adaptation: for LoRA adapters, $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices constitute the overwhelming majority of parameters. Therefore, by freezing the $\boldsymbol{A}, \boldsymbol{B}$ matrices and only updating $\boldsymbol{Z}$, we can significantly reduce the parameter cost when transferring knowledge to a new task.

Adapter Average Fine-Tuning (MHR-μ) To assess the importance of the routing parameters during few-shot adaptation, we propose an additional variant of MHR, MHR-μ, which shares the same multi-task pre-training procedure as MHR, but for each test task τ, fixes $z_\tau = (1/|\mathcal{M}|, \ldots, 1/|\mathcal{M}|)$ during few-shot adaptation. This is equivalent to discarding the routing parameters and averaging the module parameters into a single one before fine-tuning. Specifically, the adapter used during fine-tuning is initialized with (similarly for $\boldsymbol{B}^\tau$):

$$\boldsymbol{A}^\tau = \frac{1}{|\mathcal{M}|} \sum_i \boldsymbol{A}_i^*; \; \boldsymbol{A}^\tau \in \mathbb{R}^{d \times r} \qquad \text{(MHR-}\mu\text{)}$$

where $\boldsymbol{A}_i^*$ are the parameters of the adapters after MHR multi-task pre-training. Note that, differently from MHR, MHR-μ fine-tunes the same amount of parameters as the single adapter baseline. Thus, any difference in performance between the single adapter baseline and MHR-μ comes from differences in the adapter initialization and must be due to the optimization process taking place in the multi-task pre-training, before few-shot adaptation.

Routing Granularity In the original Poly, Ponti et al. [2023] showed that learning a routing matrix $\boldsymbol{Z}$ for each model layer gave better performance than sharing a single $\boldsymbol{Z}$ matrix across all layers. We therefore investigate whether this holds true also for its multi-head counterpart, MHR. In addition, we explore intermediate approaches between one $\boldsymbol{Z}$ per layer and a single one shared for the entire model. In particular, we

consider sharing $\boldsymbol{Z}$ 1) for the adapter layers belonging to the same Transformer block; or 2) for every block of l layers, which enables us to easily trade off expressivity for parameter efficiency. As we will demonstrate in section 8.5.1, this is an efficient mechanism to navigate this Pareto front in regimes of very small budgets of parameters per task.

8.4 Experiments

Our experimental evaluation aims to answer three research questions: 1) Does the expressivity of the routing function matter? 2) Why do routing-based PEFT methods yield superior performance? 3) Is routing useful during both multi-task pre-training and few-shot adaptation? We first present the baselines and datasets used in our evaluation and then discuss each question in turn.[1]

8.4.1 Baselines

In addition to `Poly`, we compare `MHR` to the following baselines for task-level generalization.

`LoRA`/`(IA)`[3] trains a single adapter common to all pre-training tasks, which is then fine-tuned on each test task separately. This is arguably the most widespread approach for parameter-efficient cross-task generalization [Liu et al., 2022, Pfeiffer et al., 2023].

`AdapterSoup` Chronopoulou et al. [2023] trains a different adapter for each task. The method only averages the adapter weights of the training tasks most similar to a given test task, before proceeding with few-shot adaptation. To compute task relatedness, we measure the cosine similarity of sentence embeddings for each task averaged over their training dataset. Notably, unlike the methods proposed in this chapter,

[1] We note that all experiments were run on a single NVIDIA A100 GPU.

there is no knowledge sharing (nor interference) during multi-task pre-training as task adapters are trained independently.

8.4.2 Datasets

We test our methods on the T0 Sanh et al. [2022] evaluation suite, following the same setup as Liu et al. [2022], and `SuperNI` Wang et al. [2022c], a large-scale dataset with more than 1,600 training tasks.

T0 Tasks We follow the pre-training and fine-tuning procedure discussed in Liu et al. [2022], using hyper-parameters and losses suggested in the public codebase for T-Few.[2] All methods were tested with T5-XL Raffel et al. [2020] and T0-3B Sanh et al. [2022] as the backbone model. Crucially, T5 is simply pre-trained on (masked) language modelling, whereas T0 is further instruction tuned: in particular, the full model is fine-tuned on examples from multiple training tasks that have been augmented with task instructions. To ensure fairness for all methods, we report the median and standard deviation of the best validation accuracy for each test task across 3 seeds, when evaluated every 50 training epochs. We treat each data subset–template pair as a unique task, yielding a total of 313 tasks.

SuperNI To limit computational costs, we report the result on 20 out of 119 test tasks. Tasks were chosen at random, with the requirement that at least 300 examples were available, and were equally split into 100 training, 100 validation and 100 test examples. For every method, we perform early stopping on the validation set. We report results with Rouge-L averaged across 3 seeds. All methods use T5-XL [Raffel et al., 2020] as the backbone and not T0, as T0 training tasks and SuperNI test tasks may overlap.

[2] https://github.com/r-three/t-few

T0 Dataset	Avg. Test
Backbone T5-XL	
(IA)^3	$62.4_{0.4}$
AdapterSoup	$62.1_{1.0}$
LoRA	$66.0_{1.6}$
LoRA-big	$65.4_{0.9}$
Poly-z	$66.4_{0.3}$
Poly	$68.0_{1.0}$
MHR-z	$68.3_{0.8}$
MHR	$\underline{69.1}_{1.0}$
Backbone T0-3B	
T-Few Liu et al. [2022]	$66.2_{0.5}$
AdapterSoup	$66.1_{0.6}$
LoRA	$67.4_{0.8}$
Poly-z	$65.3_{1.0}$
Poly	$69.0_{0.8}$
MHR-z	$68.4_{1.2}$
MHR	$\underline{69.3}_{1.2}$

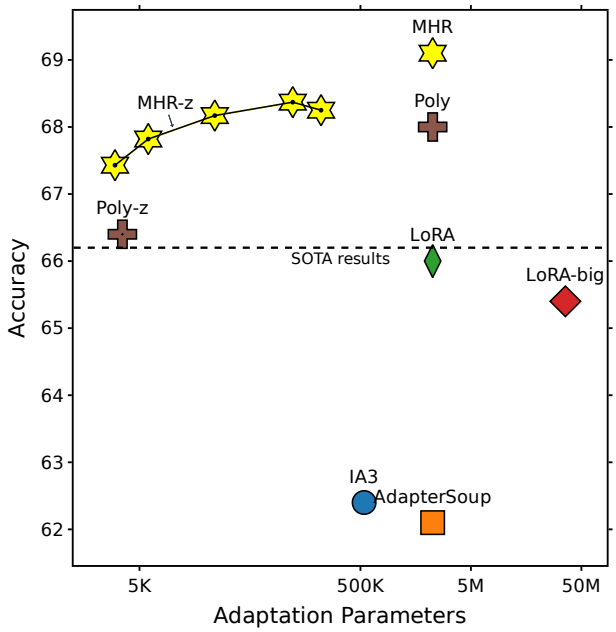

Figure 8.3 *Left:* Results of few-shot adaptation on T0 dataset Sanh et al. [2022]. We report the mean of the best validation accuracy for each test task. Subscripts correspond to standard deviation. *Right:* Accuracy of PEFT methods on the T0 dataset when applied on top of T5-XL. The x-axis shows the parameter count during the fine-tuning process.

8.5 Results and Discussion

8.5.1 Does the expressivity of the routing function matter?

MHR outperforms PEFT approaches We start our analysis by evaluating the effectiveness of our proposed technique when applied over a backbone that has not undergone prior training on instruction-following data (T5-XL). As indicated in the T0 benchmark results in the top table of Fig. 8.3, it is clear that multi-head routing techniques have a distinct advantage, outperforming both single-head routing Poly by **1.1%**, and surpassing standard LoRA approaches by an impressive **3.1%**. We also study the impact of performing instruction tuning of the full backbone before adapter training. To this end,

we also experiment with T0-3B as a backbone. In the bottom table of Fig. 8.3, we can observe that while the relative gap between MHR and baselines is smaller, multi-head routing still manages to yield favourable results. Hence, the gains of MHR compound with other multi-task methods such as instruction tuning. Finally, we turn our attention towards the SuperNI dataset (Tab. 8.2). Here, MHR continues to surpass analogous baselines.

SuperNI Dataset	Rouge-L
LoRA	$67.6_{0.8}$
LoRA-big	$67.2_{0.7}$
Poly-z	$64.6_{0.3}$
Poly	$67.8_{0.8}$
MHR-z	$68.0_{0.2}$
MHR	$\underline{68.5}_{0.3}$

Table 8.2 Results on **SuperNI** dataset. Subscripts are standard deviation.

MHR-z facilitates extreme parameter efficiency Fig. 8.3 (right) reveals intriguing findings regarding MHR-z. When we restrict training to only the routing parameters $\boldsymbol{Z}$ in the original Poly, the results are unfortunately not up to par with its version where both routing and adapters are updated. However, when we apply the same constraint to MHR, the performance is significantly closer to the optimum achieved in this setting. In fact, MHR-z surpasses prior baselines while simultaneously necessitating fewer parameters for effective adaptation to new tasks. Moreover, by controlling the number of layers which share the same Z allocation (see sec. 8.3), MHR-z is able to trade-off performance for parameter efficiency, even surpassing Poly-z in settings with only 3K trainable parameters per test task. This trend is similarly observed in the SuperNI benchmark (Tab. 8.2), where updates restricted to the routing parameters yield performance on par with standard fine-tuning. We therefore conclude that the MHR-z represents a robust approach for achieving extreme parameter efficiency in adaptation.

Additional routing heads is more beneficial than extra modules In the original `Poly` approach, a tradeoff between capacity and parameter efficiency can be achieved by adding extra modules for each adapter layer. However, this results in a linear increase in the number of multi-task parameters, which can become impractical. To explore a more effective tradeoff, we investigate the option of adding additional routing heads instead of extra modules. Fig 8.4 (right) presents the comparison between the two approaches. It demonstrates that increasing the number of routing heads leads to better performance compared to adding more modules. Importantly, the benefit of multi-head routing is twofold: it provides increased expressivity for the model, while also maintaining parameter efficiency. This finding highlights the advantage of multi-head routing as a more effective approach for balancing expressivity and parameter count in few-shot adaptation scenarios.

Routing-based methods also excel at the 11B scale We proceed to evaluate if `Poly` and `MHR` surpass established PEFT approaches when trained over a larger model backbone. To accomplish this, we employ the 11B version of T0. As depicted in Tab. 8.3, routing-based methods once again outshine standard adapter training, surpassing our reproduction of the previous state-of-the-art in Liu et al. [2022] by over 2%. We observe that `Poly` and `MHR` show similar performance in standard fine-tuning, but `MHR` z-tuning remains more performant in routing-only fine-tuning. Indeed, `MHR-`z (221K params) outperforms `Poly-`z (3.5K params) by 2.9%, while still remaining more parameter efficient than Liu et al. [2022] (1.1M params).

T0 Dataset	Avg. Test
Backbone T0-11B	
`T-Few` Liu et al. [2022]	$72.5_{0.9}$
`LoRA`	$72.3_{1.0}$
`Poly-`z	$70.0_{0.6}$
`Poly`	$\underline{74.9}_{0.6}$
`MHR-`z	$72.9_{0.8}$
`MHR`	$74.7_{0.6}$

Table 8.3 Few-shot results over 11B parameter backbones.

8.5.2 Why do routing-based PEFT methods yield superior performance?

While our proposed methods have demonstrated promising results across a broad spectrum of datasets and varying adaptation parameter budgets, the question of *why* routing-based PEFT exhibits superior performance remains unanswered. In this section, we aim to uncover the key components that drive `MHR`'s superior performance.

Learning the Routing Function is essential Given that `Poly` and `MHR` have access to more parameters than standard adapters during multi-task pretraining, we investigate whether this, and not the routing mechanism, is responsible for their superior performance. To do so, we compare them to a baseline approach. Instead of learning the routing function, we randomly assign a binary module allocation to each data point in a minibatch, disregarding task information. This random routing approach, akin to Wang et al. [2022b], allows us to directly assess the influence of additional parameters during multi-task training. At test time, the learned modules are averaged into a single one before fine-tuning; we therefore refer to this baseline as `Random`-μ. On the T0 benchmark with the T5-XL backbone, `Random`-μ performs similarly to a standard LoRA adapter (66.0%), while `Poly` and `MHR` outperform it by **2%** and **3.1%** respectively. Therefore, we conclude that learning a routing function is crucial, and merely increasing capacity during training does not directly lead to improvements.

`MHR` fosters transfer and mitigates interference across pretraining tasks Recognizing the pivotal role of the multi-task pretraining step in bolstering `Poly`'s performance, we explore the extent of transfer and interference across training tasks. By monitoring the average gradient alignment for each task pair (in terms of cosine similarity) throughout the training process, we are able to gauge the level of positive transfer. As Fig. 8.4 (left) shows, `MHR` displays a greater degree of gradient cosine similarity across tasks compared to other PEFT alternatives, including `Poly`. This finding sug-

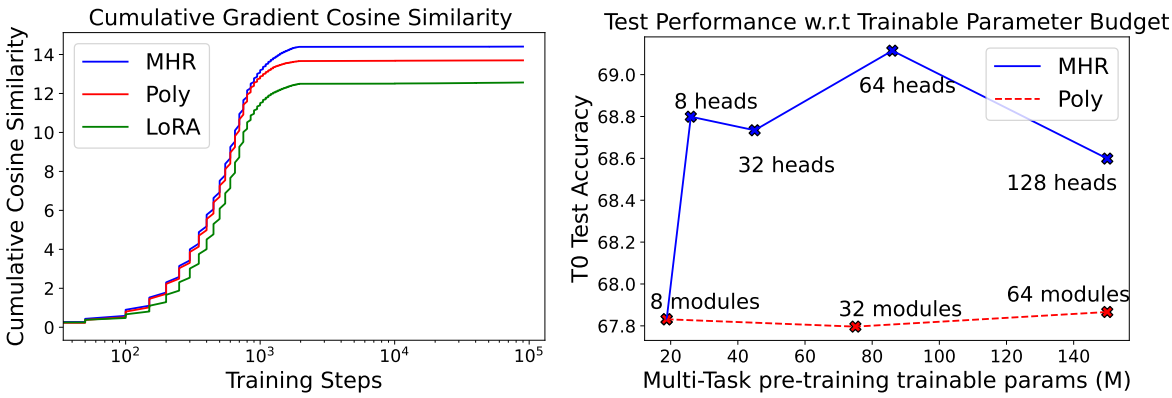

Figure 8.4 *Left:* Gradient alignment between tasks during multi-task pre-training. *Right:* Increasing the number of heads offer better scaling properties than increasing the number of modules.

gests that the enhanced flexibility offered by multi-head routing may serve to mitigate interference across tasks to a larger extent than standard routing while simultaneously promoting positive transfer.

8.5.3 Is routing important for task generalization?

We assessed the importance of routing during pre-training. We now proceed to verify whether it is important to learn routing during few-shot adaptation, too. We observe that `Poly`-μ and `MHR`-μ consistently outperform `LoRA`, and match the performance of `Poly` / `MHR` (Tab. 8.4). This demonstrates that, for few-shot adaptation, the average of the pre-trained modules provides a better initialization than learning an adapter shared across all the tasks during pre-training. The consistently superior performance of `Poly`-μ with

T0 Dataset	Test Acc.
`LoRA`	$66.0_{1.6}$
`AdapterSoup`	$62.1_{1.0}$
`Poly`	$68.0_{0.8}$
`Poly`-μ	$67.8_{0.6}$
`MHR`	$69.1_{1.1}$
`MHR`-μ	$69.1_{0.9}$
SuperNI	**Rouge-L**
`LoRA`	$67.6_{0.8}$
`Poly`	$67.8_{0.8}$
`Poly`-μ	$68.3_{0.5}$
`MHR`	$68.5_{0.6}$
`MHR`-μ	$68.5_{0.8}$

Table 8.4 Evaluating the impact of modular adaptation at test time.

respect to Random-μ and AdapterSoup stresses the importance of routing during multi-task pre-training (but not during adaptation), which provides an effective adapter initialization for few-shot learning. This finding could potentially inspire future work for improving meta-learning and weight-averaging approaches [Izmailov et al., 2018].

MHR-μ excels at zero-shot learning For many downstream tasks of interest, additional labelled data may not be available. In such settings, it is unclear how to leverage MHR-μ and Poly-μ methods. To address this, we fine-tune the average of the multi-task trained adapters on the multi-task pre-training data (instead of using the downstream few-shot data), for an additional k steps. The results are presented in Table 8.5. We find that without any additional fine-tuning ($k = 0$), averaging the adapters does not yield good results. This is due to a potential mismatch between adapters learned via task-specific routing, and the uniform routing strategy. We can observe that when fine-tuning the average of the adapters on the multi-task pre-training data for an additional k steps, MHR-μ show strong performance when evaluated in a zero-shot manner. For a fair comparison, we also additionally fine-tune LoRA for the same number of additional steps. Our best model achieves a zero-shot performance of 64.5 on top of T0-11B, achieving an absolute gain of 3.5% accuracy points.

8.6 Related Work

Multi-task learning is effective for low-resource tasks [Wei et al., 2022, Aribandi et al., 2022, Sanh et al., 2022], as knowledge can be borrowed from similar tasks by sharing the model parameters. Multi-task learning has also been applied across languages and modalities [Ponti et al., 2019, Bugliarello et al., 2022]. In the context of NLP, several families of methods enable learning new tasks from a limited set of labelled exam-

Method	Zero-Shot Test with k-shot Extra Training			
	$k = 0$	$k = 1000$	$k = 5000$	$k = 10000$
Backbone T5-XL-LM	43.2			
LoRA	56.5	56.0	56.1	55.7
Poly-μ	46.0	53.0	56.8	56.3
MHR-μ	48.0	58.0	57.1	56.3
Backbone T0-11B [Sanh et al., 2022]	61.0			
LoRA	61.2	61.6	61.5	61.5
Poly-μ	62.1	63.6	63.9	64.4
MHR-μ	63.5	64.5	64.5	64.4

Table 8.5 Zero-shot performance for MHR and the baselines, reported as the average over the 11 evaluation datasets from Sanh et al. [2022]. To obtain these zero-shot results, we average the learnt Poly/MHR adapters before performing k additional fine-tuning steps on the multi-task pretraining data. This effectively enables zero-shot transfer to downstream tasks using the same amount of parameters/flops as the baseline LoRA. MHR outperform baseline LoRA by up to 3% absolute accuracy points on T0-11B.

ples. Few-shot in-context learning [ICL; Brown et al., 2020b], where examples of a new task are concatenated into an input prompt, enables models to generalize to *unseen* tasks without any gradient-based training. Such approaches are however sensitive to the prompt format and example ordering [Zhao et al., 2021]. More importantly, ICL methods incur a significant compute overhead, as for every prediction, the full set of examples must be processed by the model [Liu et al., 2022]. To remedy this, many parameter-efficient fine-tuning (PEFT) methods have been proposed as an alternative to ICL, where a small number of new parameters are added over the frozen pretrained network. To name a few, LoRA [Hu et al., 2022] injects learnable low-rank matrices into each Transformer layer. Alternatively, the learnable matrix can be sparse, selecting nonzero shifts via the Lottery-Ticket hypothesis [Ansell et al., 2021] or via their approximate Fisher information [Sung et al., 2021]. Finally, prefix-tuning methods [Li and Liang, 2021] prepend learnable embeddings to the input or intermediate representations to specialize the model towards a downstream task.

Modular networks partition their parameters into several expert modules, each of them specialized to handle specific sub-tasks [Jacobs et al., 1991, Kirsch et al., 2018]. Modular networks are an appealing solution to the problem of adapting to unseen tasks [Corona et al., 2021], as the model can leverage its existing modules and recombine them in a novel way, thus achieving systematic generalization [Bahdanau et al., 2019]. They have also been tested in learning scenarios with data presented sequentially [Ostapenko et al., 2021], and with changing environments Goyal et al. [2021]. In NLP, mixture-of-experts (MoE) models [Shazeer et al., 2017, Fedus et al., 2021], where a learned gating mechanism routes token representations to appropriate experts (Feed-Forward layers), have shown success in scaling the number of parameters while retaining time efficiency. This results in higher performance when compared to their dense counterparts using a similar compute budget.

8.7 Discussion

In this chapter, we tackled the challenge of generalizing to new tasks based on a few examples after multi-task pre-training. Specifically, we focused on Polytropon [Ponti et al., 2023], a model where each task is associated with a subset of adapters by a routing function. We investigated how varying the level of control afforded by the routing function impacts performance on two comprehensive benchmarks for multi-task learning, T0 and Super-Natural Instructions. First, a newly proposed variant of the routing function, where multiple heads are responsible for different subsets of input dimensions, improves consistently over all other baselines, including LoRA and $(IA)^3$ adapters. Second, we identified the cause of the success of routing in its ability to prevent interference between tasks, as it yields a better alignment between their gradients. Third, we found that simple averaging of all multi-task pre-trained adapters before few-shot adaptation to new tasks provides comparable performance, thus offer-

ing state-of-the-art performance for single-adapter few-shot learning. Multi-head routing demonstrates the importance of fine-grained adapter selection for sample-efficient generalization and holds promise to improve other modular methods, such as Mixtures of Experts [MoEs; Fedus et al., 2021] in future research.

This work in this chapter, however, is limited in scope. First, the experimental procedure only considered natural language tasks. Given that adapters are also used in vision tasks for parameter efficient adaptation [He et al., 2022], an interesting future direction would be to explore whether routing based methods behave similarly in such settings. Moreover, a similar argument can be made for DNN architectural diversity; in the paper only encoder-decoder transformer model were used as backbone. Given that a considerable amount of pretrained models use decoder-only architectures [Zhang et al., 2022, Radford et al., 2019], confirming these findings with such architectures would strengthen the presented analysis. Lastly, there have been many works using PEFT adapters for continual learning [Wang et al., 2022d, Ermis et al., 2022, Smith et al., 2023]. While the work presented in this chapter did not tackle long sequences of adaptation phases, interesting follow-up work could see how to adapt `MHR` to such settings.

Chapter 9

Conclusion

In this thesis, we investigate the many facets of *efficiency* in Continual Learning. As early work in the field used replay-based methods to mitigate catastrophic forgetting, in the first chapters we optimize the *sample* efficiency of the replay step, as well as the *memory* efficiency of the replay storage. We further explore the underlying causes of forgetting, enabling us to prevent disruptive updates. Next, we shed light on the *compute* (in-)efficiency of previously proposed methods, and design new compute aware methods for forgetting prevention. In the second part of this thesis, we double down our focus on sequential knowledge acquisition under strict compute budgets, investigating how, and when to (re-)train models. Lastly, we focus on *parameter* efficient adaptation from pretrained models, enabling *sample* efficient learning to new tasks through recomposition of existing skills. In essence, my work attempts to highlight the vast array of continual learning settings, ensuring that their respective constraints and objectives are met.

9.1 Summary of Contributions

To summarize the major findings in this thesis:

- **Previous knowledge is not forgotten equally when learning new concepts.** In Chapter 4, we question whether randomly selecting points for replay is optimal. We propose a new replay sample selection criteria, which retrieves points (stored in a buffer or generated) that will be most negatively impacted by the foreseen parameter update on the incoming data. We show that it produces consistent gains in performance and greatly reduces forgetting under bounded compute constraints. Finally, we investigate the use of autoencoders to enable more efficient storage in online settings.

- **Quantization enables robust, backward-compatible representations.** In Chapter 5, we continue our investigation for more efficient storage in online settings via autoencoders. We highlight the need for backward-compatible storage of representations, to ensure that future decoder states can reconstruct representations from earlier encoder states. We show that a quantization bottleneck enables this, without removing the autoencoders's ability to adapt to changing distributions. By progressively increasing the compression rate over time through architectural changes, we can enable fully online training in a robust manner. We show that for a fixed memory budget, this yields significant gains compared to standard compression algorithms across many settings and data modalities.

- **Abrupt representation shifts cause significant forgetting.** In Chapter 6, we highlight a failure mode of replay methods in online class-incremental settings. When new classes are introduced, a sudden drop in performance is observed, irrespective of the buffer size, from which the model may or may not recover. We show that this phenomenon is caused by class prototypes of older classes being

abruptly shifted, due to newer class representations severely overlapping with older ones. Armed with this insight, we propose a new asymmetrical update rule, whereby new classes are learned in isolation, shielding previously learned representations from disruptive updates. Finally, we propose a new evaluation protocol which monitors anytime performance, as well as computation cost, and show that our methods significantly outperforms prior work, for the same compute budget.

- **When to adapt a model is setting dependent.** In Chapter 7, we depart from the class-incremental setup from previous chapters, and investigate the setting where data arrives sequentially over time in large chunks. Practitioners then have to decide how to allocate their computational budget in order to obtain the best performance at any point in time. We first formalize this learning setting in which each data chunk is drawn from the same underlying distribution, allowing us to disentangle the challenges induced by a changing distribution from the ones of learning sequentially. Central to our investigation is the question of *when* to retrain a model; fine-tuning it as soon as new data arrive improves performance in the short term, but converges to a suboptimal solution over time. Through an extensive empirical evaluation, we find that the optimal wait time depends on the difficulty of the problem and the time horizon of the full stream. Moreover, we find that having more capacity in the initial training phases, rather than increasing capacity over time as more data is observed, yields better results. Overall, we confirm that even without distribution shifts, sequential learning still poses several unaddressed challenges.

- **Expressiveness of routing functions in modular methods aids transfer.** Finally, in Chapter 8 of this thesis, we explore how to adapt LLMs to new tasks in a parameter and sample efficient way. We show that modular methods excel at this

task, by re-combining previously learned modules and fine-tuning them. We further investigate the role of the routing function in such settings. We propose a more expressive router, enabling module recomposition at a more fine-grained level. This not only yields better few-shot adaptation performance, but also enables extreme parameter efficiency by freezing the existing modules and learning only the routing function for a new task. Lastly, we show that the performance gains are due to easier optimization during the initial learning of the modules, and that one can collapse the learned modules prior to few-shot adaptation without any performance loss.

9.2 Perspective

In this section, we begin by addressing the overall limitations of the research conducted in this thesis. The goal of this thesis is to make progress towards enabling sequential knowledge acquisition in models, such that they can be reliably deployed and autonomously acquire new skills. So far, the results presented assume that the evaluation protocols, from the benchmarks to the metrics used, are a good proxy for this long-term goal. Unfortunately, the tools for evaluation at our disposal are not perfect, and limit the reach of the proposed methods. Specifically, in the first chapters, we heavily rely on class-incremental learning benchmarks as a mean for evaluating continual learning algorithms. There are indeed many aspects of these benchmarks which raise questions on whether our best performing algorithms can generalize to more diverse and realistic settings, e.g. beyond images or supervised learning. In Chapter 4, the main limitation of the proposed Maximally Interfered Retrieval (MIR) method is the lack of proper monitoring of computation during training. While it was shown that the reported gains hold even when increasing the compute budget of baselines, MIR requires additional forward-backward passes through the model, which increases

the computational cost. In Chapter 5, the primary limitation is the assumption that storage, rather than computation, is an essential bottleneck in developing Continual Learning algorithms. With the rise of large, general-purpose foundation models, the cost of training has become more significant than the expenses related to storing data and model parameters. In Chapter 6, the main limitation is that the findings on abrupt representation drift are specific to the benchmarks and settings used in the previous chapters. The relevance of this phenomenon in realistic continual learning scenarios remains uncertain. Moreover, other tasks, such as generative modeling, may not be affected by a similar phenomenon. For example, in the case of online compression discussed in Chapter 5, forgetting in the auto-encoder was gradual rather than "catastrophic." This growing body of evidence suggests that benchmarks and metrics used in Continual Learning research may not accurately reflect real-world CL use-cases, highlighting the need for more diverse and realistic evaluation methods. In Chapter 7, the primary limitations of the analysis are the significant impact of the first training run on the cumulative error rate and the focus on settings where mega-batches are drawn from the same underlying distribution. Given that across many modalities, there exist foundation models which can serve as a good initialization, leveraging such pretrained weights could enable practitioners to use longer wait times and potentially improve results. Moreover, the analysis in this chapter did not explore the impact of distribution shifts over the mega-batches, which is an important consideration in real-world scenarios. As a result, the findings may not be directly applicable to other learning settings with non-stationary distributions. Finally, in Chapter 8, the main limitation is the scope of the experimental procedure, which only considered natural language tasks. Investigating routing-based methods in vision tasks and diverse deep neural network architectures would strengthen the analysis. Furthermore, the work did not tackle long sequences of adaptation phases, which could be an interesting direction for future research.

Despite the limitations mentioned above, the research conducted in this thesis has provided valuable insights into various aspects of efficiency in continual learning. The findings can serve as a foundation for future research and development of more robust and efficient sequential learners. As the field of continual learning continues to evolve, addressing the limitations and exploring new research directions will be crucial for advancing our understanding of how AI systems can adapt and learn efficiently in dynamic environments.

9.3 Future Work

9.3.1 Merging model updates for sequential knowledge acquisition

In order to design models that can continually improve, we can greatly benefit from the ability to combine two models into a single one possessing the abilities of both models. Doing so effectively makes any transfer learning also backward compatible; merge the pretrained and finetuned model into a single, versatile one. Recent work has shown this to be possible by simply averaging parameters [Wortsman et al., 2022b]. Follow-up has shown that one can repeat this process for short task sequences with little performance degradation [Ilharco et al., 2022]. These results raise several promising research questions. What is the best representation for merging models ? Is parameter (as opposed to e.g. model activations) the right level of abstraction for model merging ? Moreover, under which conditions does model merging work (or fail) ? Can we modify the learning procedure to obtain models that are easier to merge ? One way to do so could be via meta-learning [Javed and White, 2019], which has been used to learn representations less prone to forgetting when updated sequentially. When inspecting the representations, it was found that they were highly sparse and orthogonal. Can we adapt this line of work to come up with sparse model updates which can be more easily merged without interference ? Lastly, work in model merging has mostly looked

at merging two models which lie in the same linear basin (as is often the case for multiple runs finetuned from the same checkpoint). How could one merge models which are not linearly connected ? Can the recent work in [Ainsworth et al., 2023], which proposes an algorithm that permutes activations at each layer to align two models, be used for model merging ? Lastly, can we design merging algorithms for models with different architectures ? Again, merging in the activation space could be a potential solution to this question, as it relaxes the need for identical architectures.

9.3.2 Addressing the loss of plasticity in sequential optimization

Plasticity, the ability of a model to absorb new knowledge, has been shown to degrade as training progresses [Ash and Adams, 2020]. In standard DNN training, where there is only a single training phase comprising i.i.d data, this is not a significant issue because the model can access the full data distribution at initialization, when it is plastic. On the other hand, in Continual Learning, because new data is made available over time, each learning episode becomes more challenging because of this lack of plasticity [Dohare et al., 2021]. This is issue is also prevalent in Reinforcement Learning [Lyle et al., 2023], where the data distribution naturally evolves during training from the agent's changing policy. While some progress has been made in understanding this phenomenon and designing architectures which are more robust to this issue [Sokar et al., 2023], a rigorous analysis of the significance of this phenomenon in standard transfer learning scenarios remains to be done. Specifically, when starting from pretrained models, how does the size of the backbone influence plasticity ? Can we design better pretraining (or finetuning) methods which conserve plasticity ? Leveraging meta-learning approaches to learn an update rule which favors plasticity retention could be a promising direction.

Bibliography

Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. Ss-il: Separated softmax for incremental learning. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 844–853, 2021.

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CQsmMYmlP5T.

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

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3366–3375, 2017.

Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval. In *Advances in Neural Information Processing (NeurIPS)*, 2019a.

Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019b.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019c. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/e562cd9c0768d5464b64cf61da7fc6bb-Paper.pdf.

Rahaf Aljundi, Lucas Page-Caccia, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, and Min Lin. 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*, volume 32. Curran Associates, Inc., 2019d. URL https://proceedings.neurips.cc/paper/2019/file/15825aee15eb335cc13f9b559f166ee8-Paper.pdf.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/6fb52e71b837628ac16539c1ff911667-Paper.pdf.

Alan Ansell, Edoardo Maria Ponti, Jonas Pfeiffer, Sebastian Ruder, Goran Glavaš, Ivan Vulić, and Anna Korhonen. MAD-G: Multilingual adapter generation for efficient cross-lingual transfer. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4762–4781, November 2021. doi: 10.18653/v1/2021.findings-emnlp.410. URL https://aclanthology.org/2021.findings-emnlp.410.

Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting*

of the Association for Computational Linguistics, 2022. URL https://arxiv.org/pdf/2110.07560.pdf.

Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q. Tran, Dara Bahri, Jianmo Ni, Jai Gupta, Kai Hui, Sebastian Ruder, and Donald Metzler. Ext5: Towards extreme multi-task scaling for transfer learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=Vzh1BFUCiIX.

Akari Asai, Mohammadreza Salehi, Matthew E Peters, and Hannaneh Hajishirzi. Attempt: Parameter-efficient multi-task tuning via attentional mixtures of soft prompts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6655–6672, 2022.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in Neural Information Processing Systems*, 33:3884–3894, 2020.

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks. *arXiv preprint arXiv:1802.03875*, 2018.

Ali Ayub and Alan Wagner. {EEC}: Learning to encode and regenerate images for continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=lWaz5a9lcFU.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkezXnA9YX.

Yogesh Balaji, Mehrdad Farajtabar, Dong Yin, Alex Mott, and Ang Li. The effectiveness of memory replay in large scale continual learning. *arXiv preprint arXiv:2010.02418*, 2020.

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2016.

Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O. Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 992–1001. IOS Press, 2020. doi: 10.3233/FAIA200193. URL https://doi.org/10.3233/FAIA200193.

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*, 2019.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brun-

skill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Lorenzo Bonicelli, Matteo Boschini, Angelo Porrello, Concetto Spampinato, and Simone Calderara. On the effectiveness of lipschitz-driven rehearsal in continual learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=TThSwRTt4IB.

Matteo Boschini, Lorenzo Bonicelli, Pietro Buzzega, Angelo Porrello, and Simone Calderara. Class-incremental continual learning into the extended der-verse. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020a.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020b.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Emanuele Bugliarello, Fangyu Liu, Jonas Pfeiffer, Siva Reddy, Desmond Elliott, Edoardo Maria Ponti, and Ivan Vulić. IGLUE: A benchmark for transfer learning across modalities, tasks, and languages. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 2370–2392. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/bugliarello22a.html.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

L. Caccia, H. v. Hoof, A. Courville, and J. Pineau. Deep generative modeling of lidar data. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5034–5040, 2019.

Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptive quantization modules. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1240–1250. PMLR, 13–18 Jul 2020a. URL https://proceedings.mlr.press/v119/caccia20a.html.

Lucas Caccia, Rahaf Aljundi, Nader Asadi, Tinne Tuytelaars, Joelle Pineau, and Eugene Belilovsky. New insights on reducing abrupt representation change in online continual learning. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=N8MaByOzUfb.

Lucas Caccia, Edoardo Ponti, Lucas Liu, Matheus Pereira, Nicolas Le Roux, and

Alessandro Sordoni. Multi-head adapter routing for data-efficient fine-tuning. *arXiv preprint arXiv:2211.03831*, 2022b.

Lucas Caccia, Jing Xu, Myle Ott, Marc'Aurelio Ranzato, and Ludovic Denoyer. On anytime learning at macroscale. In Sarath Chandar, Razvan Pascanu, and Doina Precup, editors, *Conference on Lifelong Learning Agents, CoLLAs 2022, 22-24 August 2022, McGill University, Montréal, Québec, Canada*, volume 199 of *Proceedings of Machine Learning Research*, pages 165–182. PMLR, 2022c. URL https://proceedings.mlr.press/v199/caccia22a.html.

Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Page-Caccia, Issam Hadj Laradji, Irina Rish, Alexandre Lacoste, David Vázquez, et al. Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems*, 33:16532–16545, 2020b.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

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

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. *arXiv preprint arXiv:1801.10112*, 2018.

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*, 2019a.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019b.

Hung-Jen Chen, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Mitigating forgetting in online continual learning via instance-aware parameterization. *Advances in Neural Information Processing Systems*, 33, 2020a.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020b.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai,

Marie Pellat, Aitor Lewkowycz, Erica Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. 2022.

Alexandra Chronopoulou, Matthew E Peters, Alexander Fraser, and Jesse Dodge. Adaptersoup: Weight averaging to improve generalization of pretrained language models. *arXiv preprint arXiv:2302.07027*, 2023.

Joseph Paul Cohen, Lan Dao, Karsten Roth, Paul Morrison, Yoshua Bengio, Almas F Abbasi, Beiyi Shen, Hoshmand Kochi Mahsa, Marzyeh Ghassemi, Haifang Li, et al. Predicting covid-19 pneumonia severity on chest x-ray with deep learning. *Cureus*, 12(7), 2020.

Rodolfo Corona, Daniel Fried, Coline Devin, Dan Klein, and Trevor Darrell. Modular networks for compositional instruction following. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1033–1040, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.81. URL https://aclanthology.org/2021.naacl-main.81.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2019.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying for-

getting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

Ludovic Denoyer and Patrick Gallinari. Deep sequential neural networks. *EWRL*, 2015.

Shibhansh Dohare, Richard S Sutton, and A Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*, pages 86–102. Springer, 2020.

Sven Eckelmann. V2v communication lidar system and positioning sensors for future fusion algorithms in connected vehicles. *Transportation Research Procedia*, 2017.

David Eigen, Ilya Sutskever, and Marc'Aurelio Ranzato. Learning factored representations in a deep mixture of experts. *ICLR*, 2014.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Beyza Ermis, Giovanni Zappella, Martin Wistuba, Aditya Rawal, and Cedric Archambeau. Memory efficient continual learning with transformers. *Advances in Neural Information Processing Systems*, 35:10629–10642, 2022.

Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

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.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135, 2017. URL https://dl.acm.org/doi/pdf/10.5555/3305381.3305498?download=true.

Genevieve Flaspohler, Francesco Orabona, Judah Cohen, Soukayna Mouatadid, Miruna Oprescu, Paulo Orenstein, and Lester Mackey. Online learning with optimism and delay. In *International Conference on Machine Learning*, 2021.

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.

Robert M French. Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. *network*, 1111:00001, 1994.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Jean-Baptiste Gaya, Thang Doan, Lucas Caccia, Laure Soulier, Ludovic Denoyer, and Roberta Raileanu. Building a subspace of policies for scalable continual learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UKr0MwZM6fL.

Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, page 0278364913491297, 2013.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

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.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=mLcmdlEUxy-.

John J. Grefenstette and Connie Loggia Ramsey. Approach to anytime learning. In *Proceedings of the Ninth International Conference on Machine Learning*, 1992.

Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

Md Yousuf Harun and Christopher Kanan. Overcoming the stability gap in continual learning. *arXiv preprint arXiv:2306.01904*, 2023.

Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*, pages 466–483. Springer, 2020.

Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

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

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

Xuehai He, Chunyuan Li, Pengchuan Zhang, Jianwei Yang, and Xin Eric Wang. Parameter-efficient model adaptation for vision transformers. *arXiv preprint arXiv:2203.16329*, 2022.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993. URL https://proceedings.neurips.cc/paper_files/paper/1993/file/9e3cfc48eccf81a0d57663e129aef3cb-Paper.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International workshop on similarity-based pattern recognition*, pages 84–92. Springer, 2015.

Christopher J Honey, Ehren L Newman, and Anna C Schapiro. Switching between

internal and external modes: a multiscale learning principle. *Network Neuroscience*, 1 (4):339–356, 2017.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chip Huyen. *Designing Machine Learning Systems*. " O'Reilly Media, Inc.", 2022a.

Chip Huyen. Real-time machine learning: Challenges and solutions. *Chip Huyen*, 2022b.

Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights. *Advances in Neural Information Processing Systems*, 2022.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *CoRR*, abs/1803.05407, 2018. URL http://arxiv.org/abs/1803.05407.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-

softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Khurram Javed and Martha White. Meta-learning representations for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/f4dd765c12f2ef67f98f3558c282a9cd-Paper.pdf.

Xu Ji, Joao Henriques, Tinne Tuytelaars, and Andrea Vedaldi. Automatic recall machines: Internal replay, continual learning and the brain. *arXiv preprint arXiv:2006.12323*, 2020.

Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4385–4393, 2018.

Michael Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 513–546, 1986.

Ellango Jothimurugesan, Ashraf Tahmasbi, Phillip B. Gibbons, and Srikanta Tirthapura. Variance-reduced stochastic gradient descent on streaming data. In *Neural Information Processing Systems*, 2018.

Pooria Joulani, Andras Gyorgy, and Csaba Szepesvari. Delay-tolerant online convex optimization: Unified analysis and adaptive-gradient algorithms. In *Association for the Advancement of Artificial Intelligence*, 2016.

Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for Transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 565–576, August 2021. URL https://aclanthology.org/2021.acl-long.47.

Atul Laxman Katole, Krishna Prasad Yellapragada, Amish Kumar Bedi, Sehaj Singh Kalra, and Mynepalli Siva Chaitanya. Hierarchical deep learning architecture for 10k objects classification. *arXiv preprint arXiv:1509.01951*, 2015.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18661–18673. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/d89a66c7c80a29b1bdbab0f2a1a94af8-Paper.pdf.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. *Advances in Neural Information Processing Systems*, 31, 2018. URL https://proceedings.neurips.cc/paper/2018/file/310ce61c90f3a46e340ee8257bc70e93-Paper.pdf.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto, technical report*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

John K Kruschke. Alcove: an exemplar-based connectionist model of category learning. *Psychological review*, 99(1):22, 1992.

John K Kruschke. Human category learning: Implications for backpropagation models. *Connection Science*, 5(1):3–36, 1993.

Matthias De Lange, Gido M van de Ven, and Tinne Tuytelaars. Continual evaluation for lifelong learning: Identifying the stability gap. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Zy350cRstc6.

Frantzeska Lavda, Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Continual classification learning using generative models, 2018.

Yann LeCun. Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models). 1987.

Yann LeCun, Leon Bottou, and and Patrick Haffner Yoshua Bengio. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Gwen Legate, Lucas Caccia, and Eugene Belilovsky. Re-weighted softmax cross-entropy to control forgetting in federated learning. In Sarath Chandar, Razvan Pascanu, and Doina Precup, editors, *Conference on Lifelong Learning Agents, CoLLAs 2023, 22-25 August 2023, McGill University, Montréal, Québec, Canada*, Proceedings of Machine Learning Research.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *CoRR*, abs/2006.16668, 2020.

Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Andrei Stoian, and David Filliat. Generative models from the perspective of continual learning. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019a.

Timothée Lesort, Alexander Gepperth, Andrei Stoian, and David Filliat. Marginal replay vs conditional replay for continual learning. In *International Conference on Artificial Neural Networks*, pages 466–480. Springer, 2019b.

Timothee LESORT, Oleksiy Ostapenko, Pau Rodriguez, Md Rifat Arefin, Diganta Misra, Laurent Charlin, and Irina Rish. Challenging common assumptions about

catastrophic forgetting, 2023. URL https://openreview.net/forum?id=LoOd40EaGA8.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(12):2935–2947, 2018. doi: 10.1109/TPAMI.2017.2773081. URL https://doi.org/10.1109/TPAMI.2017.2773081.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022. URL https://arxiv.org/abs/2205.05638.

Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2262–2268. IEEE, 2018.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Zhengying Liu, Zhen Xu, Shangeth Rajaa, Meysam Madadi, Julio C. S. Jacques Junior, Sergio Escalera, Adrien Pavao, Sebastien Treguer, Wei-Wei Tu, and Isabelle Guyon. Towards automated deep learning: Analysis of the autodl challenge series 2019. In Hugo Jair Escalante and Raia Hadsell, editors, *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, volume 123 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR, 2020.

Kuhl BA Long NM. Decoding the tradeoff between encoding and retrieval to predict memory for overlapping events. In *SSRN 3265727. 2018 Oct 13*.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6467–6476, 2017. URL http://papers.nips.cc/paper/7225-gradient-episodic-memory-for-continual-learning.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. *Proceedings of the 40th International Conference on Machine Learning*, 2023.

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single net-

work by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

Marc Masana, Xialei Liu, Bartłomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost Van De Weijer. Class-incremental learning: survey and performance evaluation on image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5513–5533, 2022.

Wojciech Masarczyk, Kamil Deja, and Tomasz Trzcinski. On robustness of generative representations against catastrophic forgetting. In *International Conference on Neural Information Processing*, pages 325–333. Springer, 2021.

James L McClelland. Complementary learning systems in the brain: A connectionist approach to explicit and implicit cognition and memory. *Annals of the New York Academy of Sciences*, 843(1):153–169, 1998.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Dulani Meedeniya, Hashara Kumarasinghe, Shammi Kolonne, Chamodi Fernando, Isabel De la Torre Díez, and Gonçalo Marques. Chest x-ray analysis empowered with deep learning: A systematic review. *Applied Soft Computing*, page 109319, 2022.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, 33:7308–7320, 2020.

Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Timothy Nguyen, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Architecture matters in continual learning. *arXiv preprint arXiv:2202.00275*, 2022.

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.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2014.

Blake Murdoch. Privacy and artificial intelligence: challenges for protecting health information in a new era. *BMC Medical Ethics*, 22(1):1–5, 2021.

Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkQqq0gRb.

Barza Nisar, Hruday Vishal Kanna Anand, and Steven L Waslander. Gradient-based maximally interfered retrieval for domain incremental 3d object detection. *arXiv preprint arXiv:2304.14460*, 2023.

Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*, 2019.

Fabrice Normandin, Florian Golemo, Oleksiy Ostapenko, Pau Rodriguez, Matthew D Riemer, Julio Hurtado, Khimya Khetarpal, Ryan Lindeborg, Lucas Cecchi, Timothée Lesort, et al. Sequoia: A software framework to unify continual learning research. *arXiv preprint arXiv:2108.01005*, 2021.

Joy D Osofsky, Howard J Osofsky, and Lakisha Y Mamon. Psychological and social impact of covid-19. *Psychological Trauma: Theory, Research, Practice, and Policy*, 12(5): 468, 2020.

Oleksiy Ostapenko, Pau Rodriguez, Massimo Caccia, and Laurent Charlin. Continual learning via local module composition. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/fe5e7cb609bdbe6d62449d61849c38b0-Abstract.html.

Oleksiy Ostapenko, Timothee Lesort, Pau Rodríguez, Md Rifat Arefin, Arthur Douillard, Irina Rish, and Laurent Charlin. Continual learning with foundation models: An empirical study of latent replay. In *Conference on Lifelong Learning Agents*, pages 60–91. PMLR, 2022.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

Lorenzo Pellegrini, Gabriele Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10203–10209. IEEE, 2020.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pages 487–503, April 2021. URL https://aclanthology.org/2021.eacl-main.39.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular deep learning. *arXiv preprint arXiv:2302.11529*, 2023. URL https://arxiv.org/pdf/2302.11529.pdf.

B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN

0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

Edoardo Ponti. *Inductive Bias and Modular Design for Sample-Efficient Neural Language Learning*. PhD thesis, University of Cambridge, 2021. URL https://aspace.repository.cam.ac.uk/bitstream/handle/1810/319303/thesis_electronic.pdf.

Edoardo Maria Ponti, Helen O'Horan, Yevgeni Berzak, Ivan Vulić, Roi Reichart, Thierry Poibeau, Ekaterina Shutova, and Anna Korhonen. Modeling language variation and universals: A survey on typological linguistics for natural language processing. *Computational Linguistics*, 45(3):559–601, 2019. URL https://watermark.silverchair.com/coli_a_00357.pdf.

Edoardo Maria Ponti, Alessandro Sordoni, Yoshua Bengio, and Siva Reddy. Combining parameter-efficient modules for task-level generalisation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 687–702, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.eacl-main.49.

Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision*, pages 524–540. Springer, 2020.

Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5822–5830, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21 (1):5485–5551, 2020.

Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Lifelong generative modeling. *arXiv preprint arXiv:1705.09847*, 2017.

Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=GhVS8_yPeEa.

Connie Loggia Ramsey and John J. Grefenstette. Case-based anytime learning. In *AAAI Technical Report WS-94-01*, 1994.

Amal Rannen, Rahaf Aljundi, Matthew B Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1320–1328, 2017.

Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285–308, 1990.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proc. CVPR*, 2017.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess,

Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=1ikK0kHjvj. Featured Certification.

Matthew Riemer, Michele Franceschini, and Tim Klinger. Generation and consolidation of recollections for efficient deep lifelong learning. *CoRR*, abs/1711.06761, 2017. URL http://arxiv.org/abs/1711.06761.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

Matthew Riemer, Tim Klinger, Djallel Bouneffouf, and Michele Franceschini. Scalable recollections for continual lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1352–1359, 2019.

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

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.

Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019. URL https://arxiv.org/pdf/1904.12774.pdf.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

David E Rumelhart. Reducing interference in distributed memories through episodic gating. *From learning theory to connectionist theory*, 1:227, 1992.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

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.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

Doyen Sahoo, Quang Pham, Jing Lu, and Steven C.H. Hoi. Online deep learning: Learning deep neural networks on the fly. In *International Joint Conferences on Artificial Intelligence Organization*, 2018.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M. Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong,

Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. Multitask prompted training enables zero-shot task generalization. In *The Tenth International Conference on Learning Representations*, 2022. URL https://arxiv.org/pdf/2110.08207.pdf.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4548–4557. PMLR, 2018.

Nan Shao, Zefan Cai, Hanwei xu, Chonghua Liao, Yanan Zheng, and Zhilin Yang. Compositional task representations for large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6axIMJA7ME3.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated

mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/pdf?id=B1ckMDqlg.

Yantao Shen, Yuanjun Xiong, Wei Xia, and Stefano Soatto. Towards backward-compatible representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6368–6377, 2020.

Dongsub Shim, Zheda Mai, Jihwan Jeong, Scott Sanner, Hyunwoo Kim, and Jongseong Jang. Online class-incremental continual learning with adversarial shapley value. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9630–9638, 2021.

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

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019.

Daniel L Silver and Robert E Mercer. The task rehearsal method of life-long learning: Overcoming impoverished data. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 90–101. Springer, 2002.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11909–11919, June 2023.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, Apr. 2020. doi: 10.1609/aaai.v34i09.7123. URL https://ojs.aaai.org/index.php/AAAI/article/view/7123.

Yi-Lin Sung, Varun Nair, and Colin Raffel. Training neural networks with fixed sparse masks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Uwh-v1HSw-x.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. *Learning to learn*, pages 3–17, 1998.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

Róbert Torfason, Fabian Mentzer, Eiríkur Ágústsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Towards image understanding from deep compression without decoding. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkXWCMbRW.

James Townsend, Tom Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. *arXiv preprint arXiv:1901.04866*, 2019.

Quang Truong, Minh Nguyen, Hy Dang, and Bo Mei. Housing price prediction via improved machine learning techniques. *Procedia Computer Science*, 174:433–442, 2020.

Chenxi Tu, Eijiro Takeuchi, Alexander Carballo, and Kazuya Takeda. Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3274–3280. IEEE, 2019.

Gido M Van de Ven, Hava T Siegelmann, and Andreas S Tolias. Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1): 4069, 2020.

Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, 2022.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Vladimir Vapnik. *Statistical learning theory*. Wiley New York, 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Eli Verwimp, Kuo Yang, Sarah Parisot, Hong Lanqing, Steven McDonagh, Eduardo Pérez-Pellitero, Matthias De Lange, and Tinne Tuytelaars. Clad: A realistic continual learning benchmark for autonomous driving. *arXiv preprint arXiv:2210.03482*, 2022.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. Exploring and predicting transferability across NLP tasks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7882–7926, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.635. URL https://aclanthology.org/2020.emnlp-main.635.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*, 2021.

Liyuan Wang, Xingxing Zhang, Kuo Yang, Longhui Yu, Chongxuan Li, Lanqing HONG, Shifeng Zhang, Zhenguo Li, Yi Zhong, and Jun Zhu. Memory replay with data compression for continual learning. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=a7H7OucbWaU.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models. *arXiv preprint arXiv:2205.12410*, 2022b.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan

Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Maitreya Patel, Kuntal Kumar Pal, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Shailaja Keyur Sampat, Savan Doshi, Siddhartha Mishra, Sujan Reddy, Sumanta Patro, Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. Benchmarking generalization via in-context instructions on 1,600+ language tasks, 2022c. URL https://arxiv.org/abs/2204.07705.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 139–149, 2022d.

Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=F1vEjWK-lH_.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 23965–23998. PMLR, 17–23 Jul 2022a. URL https://proceedings.mlr.press/v162/wortsman22a.html.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7959–7971, 2022b.

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. In *Advances in Neural Information Processing Systems*, 2020.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019.

Ju Xu and Zhanxing Zhu. Reinforced continual learning. *arXiv preprint arXiv:1805.12369*, 2018.

Le Cun Yan, B Yoshua, and H Geoffrey. Deep learning. *nature*, 521(7553):436–444, 2015.

Qinyuan Ye, Bill Yuchen Lin, and Xiang Ren. Crossfit: A few-shot learning challenge for cross-task generalization in nlp. *arXiv preprint arXiv:2104.08835*, 2021.

Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=f9D-5WNG4Nv.

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

Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task agnostic continual learning using online variational bayes. *arXiv preprint arXiv:1803.10123*, 2018.

Mengyao Zhai, Lei Chen, Fung Tung, Jiawei He, Megha Nawhal, and Greg Mori. Lifelong gan: Continual learning for conditional image generation. *ArXiv*, abs/1907.10107, 2019.

Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1131–1140, 2020.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Bowen Zhao, Xi Xiao, Guojun Gan, Bin Zhang, and Shutao Xia. Maintaining discrimination and fairness in class incremental learning. *arXiv preprint arXiv:1911.07053*, 2019.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In Marina Meila

and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/zhao21c.html.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.