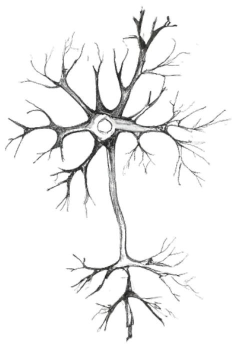
XU HE

CONTINUAL LIFELONG LEARNING IN NEURAL SYSTEMS

Overcoming Catastrophic Forgetting &
Transferring Knowledge for Future Learning



university of groningen / faculty of science and engineering / bernoulli institute

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić.

FUNDING

The research featured in this thesis has been partly supported by the European Horizon 2020 project NeuRAM3, University of Groningen, Jacobs University Bremen, Google DeepMind, and Mila - Québec AI Institute.

PRINTING

Gildeprint – The Netherlands.

university of
groningen

Continual lifelong learning in neural systems

Overcoming catastrophic forgetting and
transferring knowledge for future learning

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Thursday 4 May 2023 at 16.15 hours

by

Xu He

born on 7 June 1994
in Jiangsu, China

Supervisor

Prof. H. Jaeger

Co-supervisor

Dr. M. Sabatelli

Assessment Committee

Prof. L.R.B. Schomaker

Prof. I. Rish

Prof. A. Gepperth

Intellectual growth should commence at birth and cease only at death.

— Albert Einstein

ABSTRACT

Agents with artificial general intelligence (AGI) are supposed to learn diverse bodies of knowledge and skills over their entire lifetime. Continual learning refers to the scenarios where a machine learning system can retain and benefit from previously acquired skills while learning new ones. However, when trained on a sequence of tasks, connectionist models usually forget about previous knowledge after their parameters are adjusted for a new task. This notorious problem, known as catastrophic forgetting, poses a serious challenge towards continual lifelong learning. In this thesis, we propose novel solutions to the problem of catastrophic forgetting, continual learning and transfer learning.

- In Chapter 2, we propose a variant of the back-propagation algorithm, *Conceptor-Aided Backprop* (CAB) [He and Jaeger, 2017, 2018], in which gradients are shielded by conceptors against degradation of previously learned tasks. Conceptors have their origin in reservoir computing, where they have been previously shown to overcome catastrophic forgetting. CAB extends these results to deep feedforward networks. In particular, we apply a conceptor at each layer of the deep network to identify the linear subspace that is already used, and project the gradients to the free subspace for future learning.

- We show that when the input spaces of two tasks overlap significantly, applying CAB will lead to intransigence, in other words, the inability of a network to learn new knowledge. We analyze the reason behind it and propose another CL method in Chapter 3 called *Conceptor Pseudo-Rehearsal* (CPR) [He, 2018a] to address this limitation. Instead of projecting the weight updates on the free subspace in the pre-synaptic layer, CPR constrains them by the free subspace in the post-synaptic layer. As a result, new activations triggered by the change of parameters appear only in the free space, thus it does not interfere with old tasks. Moreover, useful features from the old tasks can be reused by the new tasks, which allows a forward transfer between tasks.

- Most continual learning approaches implicitly assume that there exists a multi-task solution for the sequence of tasks. In Chapter 4, we motivate and discuss realistic scenarios such as multi-agent games where this assumption does not hold. We

argue that the traditional metric of zero-shot remembering is not appropriate in such settings, and, inspired by the meta-learning literature, we focus on the speed of remembering previous tasks. We propose the *What and How* framework [He et al., 2020] to deal with this case by separating the concerns into what task is currently being solved and how the task should be solved. At each step, the *What* algorithm performs task inference, which allows our framework to work in absence of task boundaries. The *How* algorithm is conditioned on the inferred task, allowing for task-specific behaviour, hence relaxing the assumption of a multi-task solution. From the perspective of meta-learning, our framework is able to deal with a sequential presentation of tasks, rather than having access to the distribution of all tasks. We empirically validate the effectiveness of our approach and apply it to train generative adversarial networks (GAN).

- In Chapter 5, we study catastrophic forgetting from the perspective of information theory and define forgetting as the increase of description lengths of previous data when they are compressed with a sequentially learned model [He and Lin, 2020]. In addition, we show that continual learning approaches based on variational posterior approximation and generative replay can be considered as approximations to two prequential coding methods in compression, namely, the Bayesian mixture code and maximum likelihood (ML) plug-in code. We compare these approaches in terms of both compression and forgetting and empirically study the reasons that limit the performance of approaches based on variational posterior approximation. To address these limitations, we propose a new continual learning method called *ML Mixture Code* that combines ML plug-in and Bayesian mixture codes.
- Finally, we focus on the forward transfer aspect of continual learning and apply it to the problem of neuromorphic engineering in Chapter 6 [He, 2018b; He et al., 2019a]. Analog, unclocked, spiking neuromorphic microchips open new perspectives for implantable or wearable biosensors and bio-controllers, due to their low energy consumption and heat dissipation. However, the challenges from a computational point of view are formidable. Here we outline our solutions to realize the reservoir computing paradigm on such hardware and address the combined problems of low bit resolution, device mismatch, approximate neuron models, and timescale mismatch. The main contribution is a computational scheme, called *Reservoir Transfer*, which enables us to transfer the dynamical properties of a well-performing neural network which has been optimized on a digital computer, onto neuromorphic hardware that displays the above-mentioned problematic properties. We present a case study of implementing an ECG heartbeat abnormality detector to showcase the proposed method.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

When I started learning English, I did not expect I would completely forget how to play the bamboo flute once I become fluent in English. Why, you might wonder, would anyone forget how to play a musical instrument just because they have learned a new language? Indeed, we humans have such a remarkable ability of learning and maintaining diverse skills throughout our entire lifespan that the very thought of learning one skill will lead to dramatic forgetting of another one seems ridiculous. Thanks to the magical balance between plasticity and stability in my brain [Mermillod, Bugaiska, and Bonin, 2013; Takesian and Hensch, 2013], now I am able to write this dissertation in English, but at the same time still remember how to generate melodies with those tubes made of bamboo. However, would this be possible if my brain were a deep neural network?

The past decade has seen a sequence of successful applications of deep learning, from image recognition [Krizhevsky, Sutskever, and Hinton, 2012] to machine translation [Bahdanau, Cho, and Bengio, 2014], from reaching human-level control at playing Atari games [Mnih et al., 2015] to defeating human champion of an ancient Chinese board game [Silver et al., 2016]. Behind all of these success stories is the same training scheme: gradient descent with independently and identically distributed (i.i.d) mini-batches [Ruder, 2017]. Under this scheme, a mini-batch of data points are used to compute the gradient at every iteration of the training process, and it is assumed that the mini-batches from different iterations are samples of random variables that are independently and identically distributed. To guarantee that the i.i.d assumption holds, a common practice is to collect and store all the training data before the learning process starts, and uniformly choose a mini-batch from the entire training set at each iteration of the stochastic gradient descent algorithm. Once the training is done, the neural network is deployed for application and its parameters are supposed to stay fixed for the rest of its life.

A question that naturally arises is: what should we do when there are new data that become available? This is the very question to be addressed in this dissertation.

1.1 CATASTROPHIC FORGETTING AND CONTINUAL LEARNING

One naive strategy one may opt for when there are more data is to simply use the current parameters as initialization and restart the i.i.d gradient descent process with mini-batches of the new data. By doing so, the network is indeed able to improve its performance on the new training data. However, as McCloskey and Cohen [1989] and Ratcliff [1990] found out more than three decades ago, when a connectionist model is trained on a sequence of different datasets, the model will abruptly, if not completely, forget the information it has learned from the previous datasets once it has learned the new data. This phenomenon is called *catastrophic forgetting* (CF) or *catastrophic interference*. Catastrophic forgetting has been the major obstacle towards *continual learning* (CL) [Ring, 1998] or *lifelong learning* [Thrun, 1998]. In contrast to conventional machine learning settings, which focus on training a model to solve a single task, continual learning studies the scenario where an agent is placed in a non-stationary environment and has to learn several, distinct tasks in an arbitrary order. Ideally, the continual learner should be able to constantly absorb information from the stream of its experiences, and it should not only be able to maintain what it has learned in the past but also use its past knowledge to facilitate future learning. Continual learning is a hallmark of human intelligence and a crucial step towards artificial general intelligence (AGI). Without it, an agent will not be able to extend its repertoire of skills and adapt itself to the rapidly changing world around us.

1.2 DESIDERATA FOR CONTINUAL LEARNING

In practice, to concretely characterize what a continual learning system should achieve, the Continual Learning community had come up with a list of desiderata at the NeurIPS continual learning workshop in 2018 [*Continual learning Workshop NeurIPS 2018*; Schaul et al., 2018]:

- **Online learning**: learning takes place at all times, without predefined tasks or datasets. The data stream can change smoothly and it might not be possible to segment it by clear boundaries.
- **Forward transfer**: a continual learner should be capable of utilizing previously learned knowledge to perform better in new situations or learn faster from new tasks.
- **Backward transfer**: a continual learner should also be able to incorporate new information into its existing strategy and improve performance on tasks learned from the past.
- **Resilience to catastrophic forgetting**: learning new tasks should not dramatically decrease the performance on tasks it has already learned.
- **Bounded resources**: the agent should not expect to have infinite memory or computational power. This constraint will force the agent to use the resources

available wisely. In particular, it should be able to maximize future reward by prioritizing important knowledge and erasing redundant information.

- **YOLO**: the agent can only live once, so it should make the most out its life and cannot rely on resetting the environment or restarting its life.
- **Drinking from a firehose**: the learning agent should be able to handle a sequence of observations that is too rich to be stored in a buffer. So once an observation is gone, the agent might not have direct access to it anymore.

It is important to note that a successful continual learning algorithm does not always have to meet all the desiderata listed here. In fact, some of these desiderata can be at odds with each other. For instance, it might be impossible for a learning system with a fixed size to perfectly remember a never-ending sequence of data. After all, there is a limit to lossless data compression [Shannon, 2001]. Therefore, some continual learning algorithms might prioritize certain desiderata over the others. The right trade-offs between these potentially conflicting objectives should depend on the application scenarios and test metrics that are used to evaluate the continual learning algorithms. We discuss these aspects in the following sections.

1.3 DIFFERENT SCENARIOS OF CONTINUAL LEARNING

Since the main question continual lifelong learning studies is what the learner should do when new data show up. Depending on the type of data and the ways that the learner will be tested later, there are different scenarios of continual learning. It has been observed that some continual learning algorithms excel at a specific scenario but fail at another one. Therefore, it is hard to make meaningful comparisons between different approaches without properly categorizing and understanding these scenarios.

Conventionally, the stream of data used in continual learning is considered to be generated by a sequence of different tasks [Kirkpatrick et al., 2017; Parisi et al., 2019]. The learner is usually allowed to take its time to learn the current task before it is ready to move on to the next one. In this way, there are clear change points between different tasks, at which the learner can switch its mode to prepare itself for the next task. Focusing on this task-sequence setup, Ven and Tolias [2019] described three scenarios of continual learning based on what information about the tasks is available at testing time:

- **Task-incremental learning (Task-IL)**: this is the easiest continual learning scenario, where the learner is always told which task it is supposed to solve. Since the task information is always available, it is feasible to design a model with task-specific components. At test time, the model can then choose the component corresponding to the current task. A classical example for such architecture is the so-called "multi-head" output [Farquhar and Gal, 2019b; Zenke, Poole, and Ganguli, 2017], where the model assigns a separate output layer for each task.

- **Domain-incremental learning (Domain-IL)**: In this case, at test time the model has to solve the task at hand without the task information, but the model does not need to infer the task identity. This scenario occurs usually when the tasks in the data stream are the same but their input distributions are different.
- **Class-incremental learning (Class-IL)**: In this scenario, task information is not available at test time and the model has to infer the task identity based on the input data. This is usually considered the most difficult among these three scenarios.

One example [Ven and Tolias, 2019] used to illustrate these three scenarios is the split MNIST task protocol [Zenke, Poole, and Ganguli, 2017], in which the MNIST dataset is split into 5 parts to create 5 tasks with images of two digits in each task. The first task consists of images of zeros and ones, the second task consists of images of twos and threes, and so on. (see Table 1.1). Suppose the five tasks have already been presented sequentially to the learner for training. In the Task-IL scenario, at testing time, the learner will be given an image together with its task ID, and it has to classify this image into one of the two classes. In the Domain-IL scenario, an example test for the learner will be simply classifying an image as even or odd without knowing which task it comes from. Finally, a possible test for the Class-IL case will be asking the learner to tell which digit the image corresponds to. Note that the last question implicitly requires the learner to also infer the task ID of the image.

Task ID	1		2		3		4		5	
Digit	0	1	2	3	4	5	6	7	8	9
Class ID	1	2	1	2	1	2	1	2	1	2

Table 1.1: Five tasks created by the Split MNIST protocol. Each task contains images for two digits in the MNIST dataset, listed in the second row. Their corresponding class IDs are given in the third row. In this case, the first class corresponds to even numbers and the second class corresponds to odd numbers.

The above-mentioned categorization of scenarios is based on what task information is available at test time. At training time, however, it assumes that the task boundaries are clearly marked and task identity is available, therefore they all belong to the **task-aware** continual learning setting. There are also scenarios where the learner cannot access the task information during training, so it is not explicitly told which task is presented and when the task will change. Such scenarios are called **task-agnostic**. Many continual learning methods are not applicable in this case, since they usually rely on the task identity to select a task-specific component or require the task boundaries to decide when to consolidate already learned knowledge. In Chapter 4, we propose a continual learning framework that can be applied in the task-agnostic scenario.

1.4 TOPICS RELATED TO CONTINUAL LEARNING

Many questions studied in continual learning are closely related to other fields of Machine Learning. The techniques and theoretical tools developed in other fields have also proved to be helpful in understanding and solving problems in continual learning. For this reason, here we briefly introduce these topics and discuss their relationship to continual learning.

MULTI-TASK LEARNING Multi-Task Learning (MTL) is a machine learning framework that learns multiple related tasks at the same time. The goal is to improve overall generalization by discovering common knowledge among these tasks such that the shared representations do not overfit to a specific task. A recent survey on MTL can be found in [Zhang and Yang, 2021]. Both MTL and continual learning aim to solve multiple tasks. However MTL usually assumes that all tasks are available at the same time so that they can be trained simultaneously and the parameters of the multi-task learner are considered fixed after training, whereas in continual learning the tasks are presented sequentially and the learner should be prepared for further training at any moment.

TRANSFER LEARNING Given a target task and a source task, the goal of transfer learning is to improve the performance of the model on the target task by using the knowledge learned from the source task. Therefore, transfer learning directly relates to the forward and backward transfer aspects of continual learning, since ideally a continual learner should allow new tasks to benefit from its knowledge from the previous tasks and vice versa. However, unlike continual learning, transfer learning only focuses on the target task, it does not necessarily need to retain or improve the performance of the source task. Zhuang et al. [2020] and Chapter 4 of Sabatelli [2022] are two recent surveys on transfer learning.

META LEARNING Machine learning algorithms are usually designed by human experts to adapt the parameters of a model. Meta learning (or learning to learn) instead focuses on learning the machine learning algorithm itself based on the experience from learning related tasks. Therefore meta learning algorithms usually adapt the hyper-parameters such as the weight initialization [Finn, Abbeel, and Levine, 2017], learning rates [Li et al., 2017] or network architecture [Shaw et al., 2019]. The objective of meta learning algorithm varies depending on the applications. Examples of meta-objective includes faster adaptation [Finn, Abbeel, and Levine, 2017], better generalization [Li et al., 2018], robustness against class imbalance [Lee et al., 2020a], and so on. Due to a dramatic rise of its popularity in the past few years, there are many recent surveys available on this topic [Hospedales et al., 2020; Vanschoren, 2018]. Combining meta learning and continual learning techniques has become a promising research direction. For instance, meta learning representations or learning algorithms so that the base learner is resilient to catastrophic forgetting [Javed and White, 2019], or using meta-knowledge learned from previous tasks to accelerate learning a new task [Nagabandi, Finn, and Levine, 2019]. In Chapter 4, we propose a framework that applies meta learning to address the task-agnostic continual learning problem.

ONLINE LEARNING Online learning refers to the scenario where at each time step t, a model with parameters w_t suffers a loss l_t, which might be determined by a pair of data instances x_t, y_t. Based on this loss function, it can adapt its parameters to w_{t+1} so that its performance in the future can be improved. Here we briefly introduce two main approaches to online learning. One is based on *regret minimization*, where the regret at time T is defined as

$$R_T = \sum_{t=1}^{T} l_t(w_t) - \min_{w} \sum_{t=1}^{T} l_t(w). \tag{1.1}$$

The goal of the online learning algorithm is then to minimize the regret by choosing a sequence of parameters w_t such that as T approaches infinity, the average regret R_T/T converges to 0. This approach has its origin in online convex optimization [Hazan, 2021], so many theoretical results rely on the assumption that l_t is a convex loss function and cannot be directly applied to non-convex cases. Although closely related to continual learning, regret-based online learning does not address catastrophic forgetting and simply sidesteps this problem by storing all the data that define the losses l_t in the history. Hoi et al. [2021] wrote a comprehensive survey on this approach and its applications.

Another approach to online learning is the *Bayesian online learning* [Opper, 1998] or *streaming variational Bayes* [Broderick et al., 2013], which is based on the idea that at every time step t, the posterior $p_t(w) := p(w|x_1, y_1, \dots, x_t, y_t)$ of the model can be recursively updated by Bayes' rule

$$p_t(w) = \frac{p(x_t, y_t|w)p_{t-1}(w)}{p(x_t, y_t)}, \tag{1.2}$$

where the likelihood term $p(x_t, y_t|w)$ is the probability of the data x_t, y_t given w. Since $p_t(w)$ is intractable for most applications, online variational Bayes methods approximate it with a variational distribution $q_t(w)$ by recursively minimizing the following Kullback–Leibler (KL) divergence

$$q_t = \min_q \mathrm{KL}(q_t(w) \,||\, \frac{p(x_t, y_t|w)q_{t-1}(w)}{p(x_t, y_t)}). \tag{1.3}$$

As we will discuss later in this thesis, this idea is directly applicable in continual learning and has inspired many (online) continual learning methods.

DOMAIN ADAPTATION Domain adaptation is a special case of transfer learning where the source and target tasks are the same but their input distributions are different. For instance, in the applications of medical imaging analysis, when the scanner is suddenly replaced by another device, the MRI images might have different properties even though the tasks remain the same. The goal of domain adaptation is to adapt a model trained on the source domain to perform well on the target domain. Therefore, the forward and backward transfer problems in the domain-incremental continual learning scenario can be viewed as domain adaptation problems. Recent surveys on domain adaptation can be found in [Csurka, 2017; Wilson and Cook, 2020].

CURRICULUM LEARNING Most experiments in continual learning literature simply use a random sequence of tasks such as the permuted [Srivastava et al., 2013] or disjoint [Lee et al., 2017] MNIST tasks to train and evaluate a continual learner. Curriculum learning [Bengio et al., 2009], on the other hand, focuses on designing a sequence of tasks such that the learner can learn faster and generalize better at the end. It has been observed that a meaningful order of learning examples can boost the speed of skill acquisition for humans and animals [Krueger and Dayan, 2009], thus a more natural alternative to test continual learning algorithms can be through a well-structured curriculum. A recent work that explores this direction [Fayek, Cavedon, and Wu, 2020] showed that when the continual learner is applied in a curriculum of related and progressive tasks, it not only converged faster but also to a solution with better generalization. Similar results have also been obtained in [Luo, Kasaei, and Schomaker, 2020].

ACTIVE LEARNING Another alternative to using a random series of tasks for continual learning is to apply active learning [Settles, 2009]. Instead of letting the learner passively receive a sequence of data, active learning gives the learner the freedom to choose the next data to learn from. In practice, this is usually done by having an oracle (for example, a human annotator) in the training loop to respond to queries posed by the learner (for example, to label an unlabeled image selected by the learner). Mundt et al. [2020] and

Qureshi, Miao, and Yip [2020] showed that combining active learning and continual learning can lead to synergies of both areas and better data efficiency. Schomaker [2020] described how active label harvesting can be helpful in lifelong learning for text retrieval and recognition.

FEDERATED LEARNING In many industrial applications, data are distributed over different clients who cannot share their data with one another or upload their data to a central server. For instance, the data might be owned by different mobile device users or large organizations such as hospitals, companies or governments for whom sharing data will incur either a high communication cost or violation of privacy or security protocols [Li et al., 2021]. However, a global machine learning model trained on all of the data together can usually perform much better than a model trained on the data from a single data owner. The goal of federated learning is to a train a global model that fits all of these datasets well in a decentralized collaborative way without data leakage [Yang et al., 2019]. Numerous challenges have to be addressed in order to achieve this new learning paradigm: privacy and security challenges (providing secure computation protocols [Mohassel and Zhang, 2017] or achieving differential privacy [Geyer, Klein, and Nabi, 2018], homomorphic encryption [Acar et al., 2018; Kim et al., 2018], defenses against data poisoning [Bagdasaryan et al., 2020] and other forms of malicious attacks [Melis et al., 2019]), system challenges (overcoming communication bottleneck [Konečný et al., 2017], improving fault tolerance and handling stragglers [Smith et al., 2017]) and statistical challenges (non-IID data [Zhao et al., 2018], unbalanced distribution [Smith et al., 2017], concept drifts [Casado et al., 2020]). Among them, the statistical challenges are common to both federated learning and continual learning. Both research fields strive to obtain a global model that fits many isolated datasets generated from distinct sources. For continual learning, the isolation of datasets is temporal, in other words, learning on one dataset can only happen within a specific time window and the global model has to accumulate its knowledge over time, whereas the isolation is spatial for federated learning since learning on one dataset can only take place in a local device or server and the global model has to aggregate the knowledge across clients. Recently there have been a few works that touch upon both topics [Casado et al., 2020; Yoon et al., 2021], but little research has yet been done to thoroughly study the relationship between them.

1.5 APPROACHES TO CONTINUAL LEARNING

Since catastrophic forgetting was brought to the attention of the machine learning community, there have been many methods proposed to overcome catastrophic forgetting and to achieve continual learning [French, 1999; Goodfellow et al., 2014a]. Especially in the past five years, there is a renewed interest within the Deep Learning community to study continual learning, and this field has developed rapidly. Some common patterns have emerged from these methods, which make it possible to categorize them into different classes. In this section, we provide an overview of these methods and discuss their commonalities and distinctions. A detailed review of continual learning literature can be found in Parisi et al. [2019].

1.5.1 *Regularization-based Approaches*

These approaches alleviate catastrophic interference by imposing constraints on the update of the neural network weights. For each task, in addition to learning a model that solves the task, these methods also learn a regularization term that characterizes a region in the solution space corresponding to the current task. This regularization term can then be used to constrain the optimization process for future tasks so that new solutions are within or close to the solution region of previous tasks.

Kirkpatrick et al. [2017] proposed elastic weight consolidation (EWC), which uses the diagonal of the Fisher information matrix to weigh the importance of parameters to the learned task, and only updates the less important ones when learning a new task. Zenke, Poole, and Ganguli [2017] introduced a method called Synaptic Intelligence (SI) which maintains an online estimate of the importance of each synapse by tracking the learning trajectory. Serra et al. [2018] suggested to allocate a hard attention (HAT) mask for each neuron in the network, and jointly optimize them together with weights. As a result, these masks can indicate which neurons are already used by previous tasks, and only the weights connected with free neurons will be updated for future learning. In Chapter 2, we propose a method that uses Conceptors to characterize the linear subspace that has already been used by previous tasks, and during the back-propagation process on new tasks, the gradients are projected onto the orthogonal subspace so that the weight updates do not interfere with existing knowledge.

From a Bayesian perspective, regularization-based continual learning methods can also be seen as estimating a posterior from previous tasks and utilize it as *a prior* to regularize the learning process of new tasks, which is the same idea behind the online Bayesian learning we introduced before. Therefore, many methods in this class are based on Bayesian neural networks (BNNs) [Neal, 2012]. They differ from each other in how the posterior are parametrized and approximated.

Variational Continual Learning (VCL) [Nguyen et al., 2018] approximates the posterior of the weights by a diagonal Gaussian and use the re-parametrization trick [Kingma and Welling, 2013] to compute the gradient of the parameters of the Gaussian distribution. Ritter, Botev, and Barber [2018] achieved better approximation of the posterior with online Laplace approximation which has a block-diagonal covariance matrix based on

Kronecker factored approximation of the Hessian. Zeno et al. [2018] obtained closed form update rule for the parameters of diagonal Gaussian approximations by solving the first order necessary condition of the online objective function. This online closed-form update rule called Bayesian Gradient Descent (BGD) makes it possible to be applied directly in a task-agnostic scenario.

1.5.2 *Replay-based Approaches*

Continual learning methods that require samples in the input and output space of the network belong to this category. These samples are used to constrain the changes in the input-to-output mapping of the neural network, thus they are sometimes also called *functional* approach to continual learning. These methods differ from each other based on where these input-output samples come from.

Some replay-based methods simply use the original data stored from the past, these methods usually focus on how to select the most representative subset of the original data or how to efficiently use them. This subset of data is referred to by different names in the literature, such as memory buffer, core-set or episodic memory. Rebuffi et al. [2017] proposed a strategy called iCaRL to incrementally learn a classifier. iCaRL uses a neural network as a feature extractor and for each class, it stores a small set of exemplars and compute the mean feature of that class. Input images are classified based on the nearest-class-mean. When new classes are introduced, exemplars from previous classes are replayed to prevent the feature extractor from catastrophic forgetting. Rolnick et al. [2019] applied the idea of replay in the context of Deep Reinforcement Learning and proposed Continual Learning with Experience And Replay (CLEAR) which mixes on-policy learning from new trajectories and off-policy learning from old trajectories stored in a buffer. If the size of the buffer is bounded, they suggested to use reservoir sampling to decide when to replace the elements in the buffer by new experience when it is full. This simple method was shown to be very effective at preventing catastrophic forgetting in many Deep RL settings.

Instead of jointly training the model on the episodic memory and the new data, Gradient Episodic Memory (GEM) [Lopez-Paz et al., 2017] uses the stored samples to define inequality constraints for the optimization on the new task: the loss computed from the episodic memory cannot increase but is allowed to decrease. The motivation behind this method is to prevent the model from over-fitting the old tasks to only the stored data but at the same time still permit positive backward transfer. A follow-up work dubbed Averaged-GEM (A-GEM) was later proposed by Chaudhry et al. [2019], who showed that the quadratic programming problem in GEM can be replaced by simply computing an inner product between gradients, which makes A-GEM both computationally and memory-wise much more efficient.

Replay-based methods that do not use the original real data as the input and output samples are sometimes referred to as *pseudo-rehearsal*. This idea was first introduced in [Robins, 1995] as a solution to the problem that previously learned data are no longer available for replay. In particular, pseudo-patterns consisting of random binary input values are generated periodically during training and are fed into the network to record their corresponding output values. After some number of subsequent training iterations, previously generated pseudo-patterns are selected for rehearsal with the recorded output values as the targets. In Learning without Forgetting (LwF) [Li and Hoiem, 2017], the model consists of many task-specific parameters and a set of shared parameters. Before learning a new task, LwF allocates task-specific parameters for the new task θ_n and labels the inputs of the current task with the outputs produced by the model using the shared parameters θ_s and the parameters specific for the old task θ_o. These labels are used for rehearsal to ensure that θ_s and θ_o do not forget the old task while θ_s and θ_n are being adapted for the new task.

The effectiveness of pseudo-rehearsal is limited if the input space is large and the distribution of pseudo-inputs is very different from that of the real inputs. Thus many recent replay-based methods started to leverage advanced deep generative models to approximate the distribution of the real data. Deep Generative Replay (DGR) [Shin et al., 2017] suggests to train a generative adversarial network (GAN) in addition to a classifier. When the task switches, the samples drawn from the previous GAN can be used to approximate the loss of the old task when optimizing the current GAN and classifier. Continual Unsupervised Representation Learning (CURL) [Rao et al., 2019] is a strategy for continually learning a variational auto-encoder (VAE) with mixture-of-Gaussian latent space. The VAE itself is used for generative replay to combat its own forgetting. Pomponi, Scardapane, and Uncini [2021] used a normalizing flow (NF) to generate internal neural representations for pseudo-rehearsal. NFs are powerful invertible generative models that can efficiently perform sampling and density estimation in both directions.

1.5.3 *Expansion-based Approaches*

The defining characteristic of expansion-based approaches is that as more and more data or tasks are learned, the size of the model will grow larger. This is usually because these methods allocate a set of new parameters or a new module of the architecture for each task, thus localizing the changes to a subset of model parameters. Therefore, these approaches are sometimes also referred to as *structural, architectural or non-parametric* approaches.

Classical expansion-based methods include Growing Neural Gas (GNG) [Fritzke, 1994], which is an incremental network that uses a Hebb-like learning rule to learn topological relations among data. Since the model only adds units and connections and does not change parameters over time, it is able to learn continously over a stream of data. More recently within the field of deep learning, Rusu et al. [2016] introduced Progressive Neural Network (PNN), which blocks any change on the parameters already trained and assigns a new module for each task with lateral connection to previous modules. This prevents catastrophic forgetting by construction and allows forward transfer, at the cost of quadratic growth in model size with respect to the number of tasks. Yoon et al. [2018] proposed Dynamically Expandable Network (DEN), which selectively retrains parts of the network and dynamically extends it by adding or duplicating neurons. DEN decides how many neurons to add by group sparse regularization. Oswald et al. [2020] presented a continual learning approach based on a hyper-network [Ha, Dai, and Le, 2017] that generates the weights of a base network conditioned on task embeddings. For each task, a new low-dimension task embedding is learned together with the hyper-network to generate weights that solve the current task. Since the hyper-network is shared across all tasks, it also requires rehearsing to maintain the mapping from previous task embeddings to their corresponding task-specific weights. However, the task-specific weights do not have to be stored because they can be computed from the low-dimensional embeddings and the previous hyper-network. The Continual Neural Dirichlet Process Mixture (CN-DPM) [Lee et al., 2020b] model consists of a group of neural experts, each specializing in a subset of the data stream. Online variational inference of Dirichlet Process Mixture is applied to decide when to add more experts. Once an expert is trained on a subset of data, it is no longer updated to prevent forgetting.

1.5.4 *Meta Continual Learning*

While the continual learning methods reviewed before were all designed by human experts, there exists also works in the literature that apply meta learning techniques to continual learning problems and learn algorithms that can continually learn.

Riemer et al. [2019] defined measures of interference and transfer between two data points based on the inner product of the gradients at these two points. Based on this definition, they discussed the interference-transfer trade-off and proposed an algorithm called Meta-Experience Replay (MER), which combines experience replay with optimization based meta-learning. This method learns parameters that make future gradients less likely to cause interference but more likely to enable transfer.

Online-aware Meta-Learning (OML) [Javed and White, 2019] applies meta learning to learn a feature extractor from a distribution of tasks such that, after the meta-training phase and the feature extractor is fixed, downstream layers using representations from the feature extractor as inputs can continually learn a sequence of tasks similar to those used for meta-training. An interesting observation made by Javed and White [2019] was that although OML did not explicitly encourage sparsity, the resulting representations still turned out to be sparse, which coincides with the idea of some regularization-based continual learning methods [Golkar, Kagan, and Cho, 2019; Serra et al., 2018].

Flennerhag et al. [2020] proposed a meta learning method called Warped Gradient Descent (WarpGrad), which focuses on learning an update rule that preconditions gradients. Since gradient preconditioning can be defined point-wise in parameter space, WarpGrad can be applied in an optimization trajectory-agnostic fashion and hence scales to arbitrarily long inner loops, whereas previous optimization-based meta-learning methods are limited to few-shot setting because they require back-propagation through the inner loop optimization process. Due to this advantage, it is possible to meta-learn an optimizer for continual learning. Experiment results showed that the resulting optimizer is able to learn a series of sine regression tasks without catastrophic forgetting.

1.6 a unifying bayesian view

The approaches discussed in the previous sections appear very different from each other at first sight, but since they are all proposed to address the same problem, one may wonder if it is possible to unify them under a single framework. Having a unifying view of these approaches will not only help us understand the nature of continual learning better but can also guide us to find new approaches to continual learning. In addition, if we were to design a hybrid continual learning system that combines multiple continual learning approaches, having a unifying view makes it possible to decide a trade-off among the combinations that optimally fits a particular set of desiderata.

Farquhar and Gal [2019a] suggested that one promising candidate for unifying continual learning approaches is the Bayesian framework. From a Bayesian perspective, continual learning can be achieved simply by using the current posterior as the prior for future learning. As we discussed before, regularization-based methods fit into this framework: methods based on the Bayesian neural networks (BNNs) directly approximate the model posterior and use it as a prior for the next task; methods that do not rely on BNNs also use the Bayesian principle to justify or motivate their algorithms. Some regularization terms used by these methods can also be interpreted as Bayesian priors. Therefore, regularization-based approaches are considered *prior-focused* by Farquhar and Gal [2019a]. In contrast, replay-based methods are *likelihood-focused* since the input and output samples used for replay can be seen as approximating the likelihood terms of past tasks.

In order to clarify the roles of prior and likelihood terms in Bayesian learning objectives, we first review the concept of evidence lower bound (ELBO) or variational lower bound, which is central to many variational inference methods in machine learning. Given a dataset $D = \{(x_i, y_i)\}$ with N input-output data pairs (x_i, y_i) and a model $p(y|x, \theta)$ with parameters θ, the posterior of θ given by Bayes' theorem is $p(\theta|D) = \frac{p(D,\theta)}{p(D)}$, which is intractable to compute since the evidence $p(D) = \int_\theta p(D, \theta)$ requires an often intractable integral. Hence, variational inference methods use simpler and tractable distributions $q(\theta)$ to approximate the true posterior $p(\theta|D)$ by minimizing the KL divergence between them: $\mathrm{KL}(q(\theta)||p(\theta|D))$. Even though this KL divergence is still intractable to compute due to its dependence on the true posterior, it can be minimized since we can rewrite it in the following way:

$$\begin{aligned}
\mathrm{KL}(q(\theta)||p(\theta|D)) &= \mathbb{E}_{q(\theta)}[\log \frac{q(\theta)}{p(\theta|D)}] && (1.4)\\
&= \mathbb{E}_{q(\theta)}[\log q(\theta) - \log p(\theta, D) + \log p(D)]\\
&= \mathbb{E}_{q(\theta)}[\log \frac{q(\theta)}{p(\theta)} - \log p(D|\theta)] + \log p(D)\\
&= \mathrm{KL}(q(\theta)||p(\theta)) - \mathbb{E}_{q(\theta)}[\log p(D|\theta)] + \log p(D). && (1.5)
\end{aligned}$$

As the last term in (1.5) does not depend on θ, minimizing (1.4) is the same as minimizing the first two terms in (1.5), which are actually a negation of the ELBO:

$$\mathrm{ELBO}(q(\theta)) = -\mathrm{KL}(q(\theta)||p(\theta)) + \mathbb{E}_{q(\theta)}[\log p(D|\theta)]. \tag{1.6}$$

This provides a clear separation of the contribution from the prior $p(\theta)$ and the likelihood $p(D|\theta)$ to the objective function. It is called the evidence lower bound because (1.5) implies that

$$\log p(D) - \mathrm{ELBO}(q(\theta)) = \mathrm{KL}(q(\theta)||p(\theta|D)) \geqslant 0. \tag{1.7}$$

So $\mathrm{ELBO}(q(\theta))$ is literally a lower bound of the log evidence term $\log p(D)$. If we assume that the data pairs (x_i, y_i) with different indices i are independent and the input x_i are independent of model parameters θ, we can further rewrite the ELBO:

$$\begin{aligned}
\mathrm{ELBO}(q(\theta)) &= -\mathrm{KL}(q(\theta)||p(\theta)) + \mathbb{E}_{q(\theta)}[\log p(D|\theta)]\\
&= -\mathrm{KL}(q(\theta)||p(\theta)) + \mathbb{E}_{q(\theta)}[\sum_i^N \log p(y_i, x_i|\theta)]\\
&= -\mathrm{KL}(q(\theta)||p(\theta)) + \sum_i^N \mathbb{E}_{q(\theta)}[\log p(y_i|x_i, \theta)] + \sum_i^N \log p(x_i).
\end{aligned}$$

Again, since the last sum above does not depend on θ, maximizing the ELBO is equivalent to minimizing the following loss:

$$\mathcal{L}(q(\theta)) = \mathrm{KL}(q(\theta)||p(\theta)) + \sum_{i}^{N} \mathbb{E}_{q(\theta)}[\log p(y_i|x_i, \theta)]. \tag{1.8}$$

Now consider continually learning a sequence of T datasets $D_t = \{(x_i^t, y_i^t)\}$ for $1 \leqslant t \leqslant T$, given (1.8), training a BNN with parameters θ jointly on all datasets leads to the following loss of the posterior $q_T(\theta)$:

$$\mathcal{L}^T(q_T(\theta)) = -\sum_{t=1}^{T} \left[\sum_{i}^{N_t} \mathbb{E}_{q_T(\theta)}[\log p(y_i^t|\theta, x_i^t)] \right] + \mathrm{KL}(q_T(\theta)||p(\theta)), \tag{1.9}$$

where the first term is the sum of negative log-likelihoods on all datasets and the second term is the KL divergence between the posterior and an initial prior $p(\theta)$ chosen before any data is observed.

For continual learning, the previous datasets D_t for $t < T$ might not be available. Prior-focused methods such as VCL therefore use the following loss instead

$$\mathcal{L}_{\mathrm{VCL}}^T(q_T(\theta)) = -\sum_{i}^{N_T} \mathbb{E}_{q_T(\theta)}[\log p(y_T^t|\theta, x_T^t)] + \mathrm{KL}(q_T(\theta)||q_{T-1}(\theta)), \tag{1.10}$$

where the sum of negative log-likelihoods $-\sum_{t=1}^{T-1} \left[\sum_{i}^{N_t} \mathbb{E}_{q_T(\theta)}[\log p(y_i^t|\theta, x_i^t)] \right]$ on previous datasets and the KL divergence term $\mathrm{KL}(q_T(\theta)||p(\theta))$ in (1.9) are replaced by a KL divergence $\mathrm{KL}(q_T(\theta)||q_{T-1}(\theta))$ between the current posterior and the previous posterior $q_{T-1}(\theta)$. $q_0(\theta) := p(\theta)$ is the initial prior.

On the other hand, replay-based methods are considered likelihood-focused since they approximate the log-likelihood term of a previous dataset D_t for $t < T$ in (1.9) by a generative model $p_t(x, y)$:

$$\frac{1}{N_t} \sum_{i}^{N_t} \mathbb{E}_{q_T(\theta)}[\log p(y_i^t|\theta, x_i^t)] \approx \int \log p(y|\theta, x) p_t(x, y) q_T(\theta) dx dy d\theta. \tag{1.11}$$

In practice, since the integral in (1.11) is usually computed by Monte Carlo sampling, the generative model $p_t(x, y)$ can be implemented by storing a coreset or training a deep generative model such as a GAN or a VAE, as we have reviewed before.

Farquhar and Gal [2019a] did not provide a Bayesian interpretation of expansion-based approaches. However, there are expansion methods based on Bayesian nonparametric priors, which fit naturally to this unifying view. For example, Kessler et al. [2019], Kumar, Chatterjee, and Rai [2019], and Mehta et al. [2021] all used an Indian Buffet Process (IBP) prior to model the structure of a neural network. By online variational inference of the Beta and Bernoulli distribution that define the IBP, their models can automatically decide when and by how much to expand the model complexity.

In Chapter 5, we propose an alternative framework for unifying the prior-focused and likelihood-focused approaches based on information theory and minimum description length.

1.7 THESIS OUTLINE

In this section, we discuss the research questions studied in this thesis and provide an overview of our main contributions.

CONTINUAL LEARNING WITH CONCEPTORS *How do we quantify the capacity of a deep network? Does a deep network always exhaust its capacity after it is trained on a task? If not, how can we exploit the remaining capacity for learning new tasks?*

These questions are addressed in Chapter 2 and 3. We study the relationship between L2 regularization and continual learning. We show that when a neural network is trained with L2 regularization, which is a commonly used technique to prevent over-fitting, the neural network weights do not always use the entire input space in the previous layer. A novel neural mechanism called conceptors can be used to identify the linear subspace that is already used by the weight matrices and their orthogonal subspace can be considered still free. In order to exploit the spared capacity to learn new tasks, we propose a variant of the back-propagation algorithm called Conceptor-Aided Backprop (CAB) to project the gradients onto the free subspace in order to reduce the interference between the learning process and the already learned knowledge from the past.

In Chapter 3, we study the limitations of CAB that makes it fail in certain scenarios of continual learning. To overcome these drawbacks, we propose an alternative continual learning method that constrains the learning of weights based on the conceptors over the output space instead of the input space. The resulting method applies pseudo-rehearsal mechanism to ensure the constrained optimization.

CONTINUAL LEARNING IN TASK AGNOSTIC SCENARIO *How can we manage to continually learn a model in scenarios such as multi-agent games in RL or GANs, where the learning algorithm is agnostic of task changes and the current task may require a different solution than the previous tasks?*

In Chapter 4, we show that catastrophic forgetting is in fact inevitable when task information is unknown and different tasks are conflicting with each other. The solution is to instead learn a meta-model, and rely on task inference to quickly recover performance when the task changes. To this end, we proposed the *What and How* framework, and applied Bayesian continual learning techniques on the meta level. Experimental results show that our framework significantly outperform other alternatives in many task-agnostic scenarios and it is also shown to be effective in preventing the mode-collapse problem in GANs.

CONTINUAL LEARNING FROM THE PERSPECTIVE OF COMPRESSION *If we consider learning as the process of compressing data into a model, then what is forgetting in this view? Especially, can we formalize the concept of forgetting? Can this perspective help us understand existing continual learning approaches and even lead us to new continual learning methods?*

In Chapter 5, we study catastrophic forgetting from the perspective of information theory and define forgetting as the increase of description lengths of previous data when they are compressed with a sequentially learned model. In addition, we show that the prior-focused and likelihood-focused approaches correspond to two predictive sequential (prequential) codes: the Bayesian mixture code and maximum likelihood (ML) plug-in code. We empirically compare these approaches and discuss the reasons that limit the performance of prior-focused methods. In order to overcome these limitations, we propose a new continual learning method that hierarchically combines ML plug-in and Bayesian mixture codes.

CONTINUAL LEARNING AS KNOWLEDGE TRANSFER *How can we build a well-performing reservoir in an analog, unclocked, spiking neuromorphic microchip which has problems such as low bit resolution, device mismatch, inaccurate neuron models, and timescale mismatch?*

The theory of reservoir computing provides guidelines for how to initialize a well-performing reservoir in digital simulations. However, these guidelines cannot be directly applied when the reservoir is realized in a neuromorphic hardware with problems described above. Other alternative methods for constructing a good recurrent spiking network in such devices also impose formidable challenges from a computational point of view, because they require either calibration of individual neurons or running back-propagation through time (BPTT) on the analog unclocked microchips, which are infeasible due to the same problems listed above. In Chapter 6, we propose a simple yet effective solution to overcome these challenges. Our main contribution is a transfer learning scheme for recurrent networks, called Reservoir Transfer, which makes it possible for us to transfer the dynamical properties of a well-performing recurrent network, created and optimized on a digital computer, onto a neuromorphic hardware that displays the above-mentioned problematic properties. We showcase the proposed scheme with an application to ECG heartbeat abnormality detection.

2 CONCEPTOR-AIDED BACKPROPAGATION

The capacity of a deep neural network can be defined in multiple ways and it has been shown that neural networks with different amount of neurons and layers have different capacity [Baldi and Vershynin, 2019; Guss and Salakhutdinov, 2018; Sontag, 1998]. So does a neural network always use up its full capacity once it is trained on a task? If not, can we quantify how much capacity is used up and exploit the remaining power of the network? In this chapter, we will show that the theory of conceptors provides us with a tool to quantify capacity of a neural network and identify the used and free space within the network. In addition, we propose a variant of the back-propagation algorithm called *Conceptor-Aided Backprop* (CAB) to train deep networks. For each layer of the network, CAB computes a conceptor to characterize the linear subspace spanned by the neural activation vectors appeared in the already learned tasks. When the network is trained on a new task, CAB uses the conceptor to project the gradients to the subspace orthogonal to the used subspace so that the linear mapping already learned in that layer will be preserved after applying the gradient descent updates. The rest of this chapter is a verbatim copy of He and Jaeger [2017, 2018] with a few modifications. [1]

2.1 CONCEPTORS

Conceptors are a general-purpose neuro-computational mechanism that can be used in a diversity of neural information processing tasks including temporal pattern classification, one-shot learning, human motion pattern generation, de-noising and signal separation [Jaeger, 2017]. In this section, we review the basics of conceptor theory and its application to incrementally training linear readouts of recurrent neural networks as used in reservoir computing. A comprehensive treatment can be found in [Jaeger, 2014].

A *matrix conceptor* C for some vector-valued random variable $x \in \mathbb{R}^N$ is defined as a linear transformation that minimizes the following loss function:

$$\mathcal{J}(C, \alpha) = \mathbb{E}_x[\|x - Cx\|^2] + \alpha^{-2}\|C\|^2_{\text{fro}},$$

where α is a control parameter called *aperture* and $\|\cdot\|_{\text{fro}}$ is the Frobenius norm. This optimization problem has a closed-form solution:

$$C = R(R + \alpha^{-2} I)^{-1}, \tag{2.1}$$

1 [He and Jaeger, 2017] was also presented at the IV Workshop on Dynamical Systems and Brain-inspired Information Processing in 2017 and received the Best Poster Award.

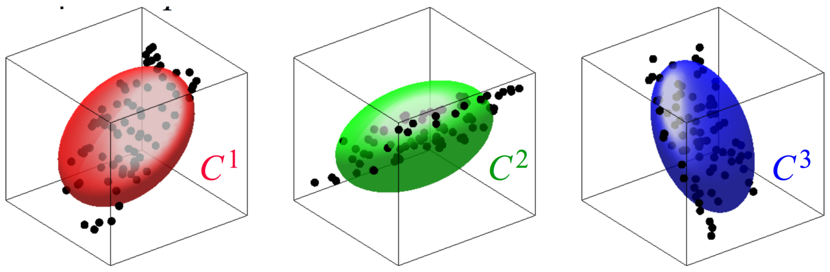


Figure 2.1: 3D point clouds (black dots) and their corresponding conceptors, represented by ellipsoids whose axes are the singular vectors of conceptors and the lengths of these axes match the singular values of conceptors. Each edge of the plot boxes range from -1 to $+1$ admitted by neural dynamics with a $\tanh$ nonlinearity; conceptor ellipsiods lie inside the unit sphere. Image taken from [Jaeger, 2014]

where $R = \mathbb{E}_x[xx^\top]$ is the $N \times N$ auto-correlation matrix of x, and I is the $N \times N$ identity matrix.

SVD OF CONCEPTORS The result given in (2.1) can be understood by studying the singular value decomposition (SVD) of C. If $R = U\Sigma U^\top$ is the SVD of R, then the SVD of C is given as $USU^\top$, where the singular values s_i of C can be written in terms of the singular values σ_i of R: $s_i = \sigma_i/(\sigma_i + \alpha^{-2}) \in [0, 1)$. In intuitive terms, C is a soft projection matrix on the linear subspace occupied by the samples of x. For a vector $\hat{x}$ in this subspace, C acts like the identity map: $C\hat{x} \approx \hat{x}$, and when some noise ϵ orthogonal to the subspace is added to y, C de-noises: $C(\hat{x} + \epsilon) \approx \hat{x}$. Figure 2.1 shows the ellipsoids corresponding to three sets of $\mathbb{R}^3$ points.

QUOTA OF CONCEPTORS We define the quota $Q(C)$ of a conceptor to be the mean singular values: $Q(C) := \frac{1}{N} \sum_{i=1}^{N} s_i$. Since s_i range between 0 and 1, $Q(C)$ is also a value between 0 and 1. Intuitively, the quota measures the fraction of the total dimensions of the entire vector space that is claimed by C.

BOOLEAN LOGIC ON CONCEPTORS Moreover, logic operations that satisfy most laws of Boolean logic can be defined on matrix conceptors as the following:

$$\neg C := I - C, \tag{2.2}$$

$$C^i \vee C^j := (R^i + R^j)(R^i + R^j + \alpha^{-2} I)^{-1}, \tag{2.3}$$

$$C^i \wedge C^j := \neg(\neg C^i \vee \neg C^j), \tag{2.4}$$

where the **negation** $\neg C$ is a linear transformation that softly projects onto a linear subspace that can be roughly understood as the orthogonal complement of the subspace characterized by C. The **disjunction** $C^i \vee C^j$ is the conceptor computed from the union of the two sets of sample points from which C^i and C^j are computed. It describes a linear subspace that is approximately the sum of linear subspace characterized by C^i and C^j, respectively. The definition of the **conjunction** $C^i \wedge C^j$ reflects the de Morgan's law. Figure 2.2 illustrates the geometry of these operations.

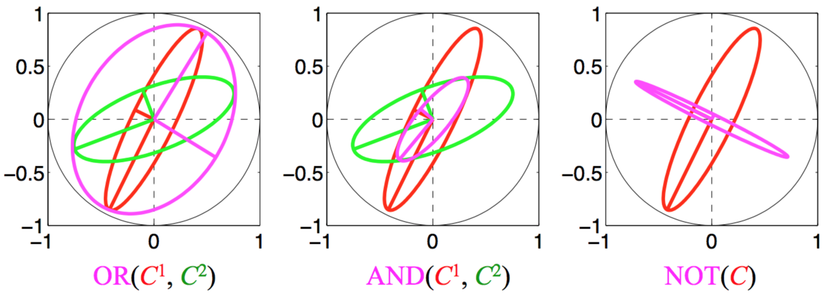


Figure 2.2: Geometry of Boolean operations on 2-dimensional conceptors. The OR (resp. AND) operation gives a conceptor whose ellipsoid approximately is the smallest (largest) ellipsoid enclosing (contained in) the argument conceptor's ellipsoids. Image taken from [Jaeger, 2014]

2.2 INCREMENTAL RIDGE REGRESSION

This section explains how conceptors can be applied to master continual learning in a simple linear model trained on a supervised task by ridge regression. The training is done sequentially on multiple input-to-output mapping tasks. This simplified scenario illustrates the working principle of continual learning with conceptors and will later be used repeatedly as a sub-procedure in the CAB algorithm for training multi-layer feed-forward networks.

Consider a sequence of m incoming tasks indexed by j. We denote the training dataset for the j-th task by $D^j := \{(x_1^j, y_1^j), \cdots, (x_n^j, y_n^j)\}$, where $x_i^j \in \mathbb{R}^N$ are input vectors and $y_i^j \in \mathbb{R}^M$ their corresponding target outputs. Training a linear model only on the dataset

D^j by ridge regression leads to a task-specific solution $\hat{W}^j$ that minimizes the following loss function

$$\hat{W}^j := \arg\min_{W} \frac{1}{n} \sum_{i=1}^{n} \|y_i^j - Wx_i^j\|^2] + \alpha^{-2}\|W\|_{\text{fro}}^2. \tag{2.5}$$

Since $\hat{W}^j$ does not depend on the dataset of any other tasks, in general there is no guarantee that $\hat{W}^j$ can fit the dataset D^k for $k \neq j$. So naively training a linear model sequentially on each task might lead to catastrophic forgetting.

We now describe a conceptor-based incremental ridge regression algorithm designed to address this problem. Whenever the training dataset for a new task is available, the incremental method computes a matrix conceptor C^j for the input variable of the new task using Equation 2.1 and updates the linear model W^j, resulting in a sequence of linear models $W^1, \ldots W^m$. Our goal is that W^j solves not only the j-th task but also all previous tasks: for $k \leqslant j$, $y^k \approx W^j x^k$. Since the conceptor C^j is a soft projection matrix onto the linear subspace spanned by input patterns from the j-th task, the disjuntion of all previous conceptors $A^{j-1} = C^1 \vee \cdots \vee C^{j-1}$ characterizes the memory space already claimed by the previous tasks $1, \ldots, j-1$. Therefore, $F^j = \neg A^{j-1}$, the orthogonal complement of $A^j - 1$, represents the memory space still free for the j-th task. Here "memory space" refers to the linear subspace of input vectors. In detail, this method proceeds in the following way:

- **Initialization (no task trained yet):** $W^0 = 0_{M \times N}, A^0 = 0_{N \times N}$.
- **Incremental task learning:** For tasks $j = 1, \ldots, m$ do:
  1. Store the input vectors from the j-th training dataset of size n into a $N \times n$ sized input collection matrix X^j, and store the output vectors into a $M \times n$ sized output collection matrix Y^j.
  2. Compute the conceptor for this task by $C^j = R^j(R^j + \alpha^{-2}I)^{-1}$, where $R^j = \frac{1}{n}X^j X^{j\top}$
  3. Train an increment matrix W_{inc}^j (to be added to W^{j-1}, yielding W^j), with the crucial aid of a helper conceptor F^j:
     (a) $F^j := \neg A^{j-1}$ (*comment: this conceptor characterizes the "still disposable" memory space for the* j*-th task*),
     (b) $T := Y^j - (W^{j-1}X^j)$ (*comment: this matrix consists of target values for a linear regression to compute* W_{inc}^j),
     (c) $S := F^j X^j$ (*comment: this matrix consists of input arguments for the linear regression*),

(d) $W^j_{\text{inc}} = TS^\top(SS^\top + n\lambda^{-2}I)^{-1}$ (*comment: carry out the regression, regularized by* λ^{-2}),

4. Update W^j: $W^j = W^{j-1} + W^j_{\text{inc}}$.
5. Update $A : A^j = A^{j-1} \vee C^j$ (*comment: this is possible due to the associativity of the* $\vee$ *operation on conceptors*)

LEARNING WITHOUT FORGETTING The weight increment W^j_{inc} in 3(d) does not interfere much with the previously learned weights W^{j-1} because the regularization in step 3(d) constrains the row space of W^j_{inc} to be only the linear subspace spanned by input arguments defined in 3(c), which are inside the kernel of W^{j-1} due to the projection by F^j. Intuitively speaking, when learning a new task, this algorithm exploits only the components of input vectors in the still unused space (kernel of W^{j-1}, characterized by F^j) to compensate errors for the new task and leaves the directions in the already used memory space (row space of W^{j-1}, characterized by A^{j-1}) intact.

Formally, we can rewrite W^j_{inc} as follows:

$$W^j_{\text{inc}} = TS^\top(SS^\top + n\lambda^{-2}I)^{-1} = T(SS^\top + n\lambda^{-2}I)^{-1}S^\top \tag{2.6}$$

$$= T(SS^\top + n\lambda^{-2}I)^{-1}X^{j\top}F^j, \tag{2.7}$$

where 2.6 can be derived by either applying the Woodbury matrix identity or the (simpler) push-through identity, and $F^j = F^{j\top}$ since F^j is symmetric. If we assume the joint conceptor A^{j-1} on all previous tasks can be considered a sharp projection matrix, in other words, if its singular values are either zeros or ones, then by the idempotence of projection matrices, we have

$$F^jA^{j-1} = (\neg A^{j-1})A^{j-1} = (I - A^{j-1})A^{j-1} = 0. \tag{2.8}$$

Combining this with (2.7), it follows that $W^j_{\text{inc}}A^{j-1} = 0$, which can be used to show that the proposed incremental ridge regression method is able to learn new tasks without forgetting old tasks.

W^j_{inc} is the closed-form solution to the following cost function

$$\mathcal{J}(W^j_{\text{inc}}) := \mathbb{E}[|W^j_{\text{inc}}s - t|^2] + \lambda^{-2}|W^j_{\text{inc}}|^2_{\text{fro}}, \tag{2.9}$$

where $t = y^j - W^{j-1}x^j, s = F^j x^j$. If W^j_{inc} can successfully achieve the objective specified in 2.9 such that $W^j_{\text{inc}} s - t \approx 0$, then the updated weights W^j can also successfully learn the data in D^j:

$$\begin{aligned} W^j x^j =&(W^{j-1} + W^j_{\text{inc}})x^j = W^{j-1}x^j + W^j_{\text{inc}}x^j \\ =&y^j - t + W^j_{\text{inc}}(A^{j-1} + F^j)x^j = y^j - t + W^j_{\text{inc}}A^{j-1}x^j + W^j_{\text{inc}}s \\ =&y^j - t + 0 + W^j_{\text{inc}}s \approx y^j. \end{aligned} \tag{2.10}$$

Furthermore, assuming the previous weights W^{j-1} can solve all tasks up to $j-1$, in other words, for $k \leqslant j-1$, $W^{j-1}x^k \approx y^k$, we can also show that this method does not forget previous tasks:

$$\begin{aligned} W^j x^k =&(W^{j-1} + W^j_{\text{inc}})x^k = W^{j-1}x^k + W^j_{\text{inc}}x^k \\ =&W^{j-1}x^k + W^j_{\text{inc}}A^{j-1}x^k = W^{j-1}x^k + 0 \approx y^j, \end{aligned} \tag{2.11}$$

where $x^k = A^{j-1}x^k$ since x^k is in the subspace A^{j-1} projects onto.

2.3 CONCEPTOR-AIDED SGD AND BACK-PROP

In this section, we first derive a stochastic gradient descent version of the algorithm described in the previous section, then present the procedure of CAB.

2.3.1 *SGD*

In the algorithm introduced in the previous section, W^j_{inc} is computed by ridge regression, which directly returns the solution for the loss function in 2.9. One can also minimize this cost function by stochastic gradient descent (SGD), which starts from an initial guess of W^j_{inc} and repeatedly performs the following update

$$W^j_{\text{inc}} \leftarrow W^j_{\text{inc}} - \eta \nabla_{W^j_{\text{inc}}} \mathcal{J}(W^j_{\text{inc}}), \tag{2.12}$$

where η is the learning rate and the gradient is given by:

$$\nabla_{W^j_{\text{inc}}} \mathcal{J}(W^j_{\text{inc}}) = 2\mathbb{E}[(W^j_{\text{inc}}s - t)s^\top] + 2\lambda^{-2}W^j_{\text{inc}}. \tag{2.13}$$

Substituting t by $y^j - W^{j-1}x^j$ and s by $F^j x^j = (I - A^{j-1})x^j$ in (2.13), we get

$$\nabla_{W^j_{\text{inc}}} \mathcal{J}(W^j_{\text{inc}}) \tag{2.14}$$

$$= 2\mathbb{E}[(W^j_{\text{inc}}(I - A^{j-1})x^j - y^j + W^{j-1}x^j)s^\top] + 2\lambda^{-2}W^j_{\text{inc}} \tag{2.15}$$

$$= 2\mathbb{E}[(-W^j_{\text{inc}}A^{j-1}x^j + (W^{j-1} + W^j_{\text{inc}})x^j - y^j)s^\top] + 2\lambda^{-2}W^j_{\text{inc}}. \tag{2.16}$$

Due to the regularization term in the cost function and the assumption that A^{j-1} can be considered a sharp projection matrix, as the optimization goes on, eventually the component in W_{inc} that is orthogonal to F^j will vanish from weight decay because all the input s are only inside the linear subspace characterized by F^j, hence component in W_{inc} orthogonal to it will make no difference in the output but only increase the Frobenius norm of W_{inc}. As a result, $W_{inc}^j A^{j-1} x^j$ will converge to 0 as the algorithm proceeds. In addition, since $W^j = W^{j-1} + W_{inc}^j$, (2.16) can be simplified to

$$\nabla_{W_{inc}^j} \mathcal{J}(W_{inc}^j) = 2\mathbb{E}[(W^j x^j - y^j)s^\top] + 2\lambda^{-2} W_{inc}^j. \tag{2.17}$$

Adding W^{j-1} to both sides of (2.12), we obtain the update rule for W^j:

$$W^j \leftarrow W^j - 2\eta\mathbb{E}[es^\top] + 2\eta\lambda^{-2} W_{inc}^j, \tag{2.18}$$

where $e := W^j x^j - y^j$. In practice, at every iteration, the expected value can be approximated by a mini-batch of size n_B, indexed by i_B:

$$\hat{\mathbb{E}}[es^\top] = \frac{1}{n_B} \sum_{i_B=0}^{L} (W^j x_{i_B}^j - y_{i_B}^j)(F^j x_{i_B}^j)^\top = \frac{1}{n_B} \sum_{i_B=0}^{L} (W^j x_{i_B}^j - y_{i_B}^j) x_{i_B}^{j\top} F^j, \tag{2.19}$$

where the transpose for F^j can be dropped since it is symmetric.

If we only train the j−th task without considering the previous tasks, the update rule given by normal SGD is

$$W^j \leftarrow W^j - 2\eta\mathbb{E}[ex^{j\top}] + 2\eta\lambda^{-2} W^j. \tag{2.20}$$

Comparing this to the update rule in (2.18), we notice two modifications when a conceptor is adopted to avoid catastrophic forgetting: first, the gradient of weights are calculated using the conceptor-projected input vector $s = F^j x^j$ instead of the original input vector x^j; second, regularization is done on the weight increment W_{inc}^j rather than the final weight W^j. These two modifications lead to our design of the conceptor-aided algorithm for training multilayer feed-forward networks.

2.3.2 *Backprop*

The basic idea of CAB is to guide the gradients of the loss function on every linear component of the network by a matrix conceptor computed from previous tasks during error back-propagation [Rumelhart, Hinton, and Williams, 1986], repeatedly applying the conceptor-aided SGD technique introduced in the previous section in every layer.

Consider a feed-forward network with $L+1$ layers, indexed by $l = 0, \dots L$, such that the 0-th and the L-th layers are the input and output layers respectively. $W^{(l)}$ represents the linear connections between the $(l-1)$-th and the l-th layer, where we refer to the former as the pre-synaptic layer with respect to $W^{(l)}$, and to the latter as the post-synaptic layer. We denote by $N^{(l)}$ the size of the l-th layer (excluding the bias unit) and $A^{(l)^j}$ a conceptor characterizing the memory space in the l-th layer used up by the first j tasks. Let $\sigma(\cdot)$ be the activation function of the nonlinear neurons and θ all the parameters of the network to be trained. Then the incremental training method with CAB proceeds as follows:

- **Initialization (no task trained yet):** $\forall l = 0, \dots, L-1$, $A^{(l)^0} := 0_{(N^{(l)}+1)\times(N^{(l)}+1)}$, and randomly initialize $W^{(l+1)^0}$ to be a matrix of size $N^{(l+1)} \times (N^{(l)}+1)$.
- **Incremental task learning:** For $j = 1, \dots, m$ do:
  1. $\forall l = 0, \dots, L-1, F^{(l)^j} = \neg A^{(l)^{(j-1)}}$. (*This conceptor characterizes the still disposable vector space in layer* l *for learning task* j)
  2. Update the network parameters $\theta^{(j-1)}$ obtained after training the first $j-1$ tasks to θ^j by stochastic gradient descent, where the gradients are computed by CAB instead of the classical backprop. Algorithms 1 and 2 detail the forward and backward pass of CAB, respectively. Different from classical backprop, the gradients are guided by a matrix conceptor $F^{(l)^j}$, such that in each layer only the activity in the still disposable memory space will contribute to the gradient. Note that the conceptors remain the same until convergence of the network for task j.
  3. After training on the j-th task, run the forward procedure again on a batch of n_B input vectors, indexed by i_B, taken from the j-th training dataset, to collect activations $h_{i_B}^{(l)^j}$ of each layer into a $N^{(l)} \times n_B$ sized matrix $H^{(l)^j}$, and set the correlation matrix $R^{(l)^j} = \frac{1}{n_B} H^{(l)^j} (H^{(l)^j})^\top$.
  4. Compute a conceptor on the l-th layer for the j-th pattern by $C^{(l)^j} = R^{(l)^j}(R^{(l)^j} + \alpha^{-2} I_{N^{(l)}\times N^{(l)}})^{-1}, \forall l = 0, \dots, L-1$. Finding an optimal aperture can be done by a cross-validation search[2].
  5. Update the conceptor for already used space in every layer: $A^{(l)^j} = A^{(l)^j} \vee C^{(l)^j}, \forall l = 0, \dots, L-1$.

2 Jaeger [2014] proposed a number of methods for analytical aperture optimization. It remains for future work to determine how these methods transfer to our situation.

Algorithm 1 The forward procedure of conceptor-aided backprop, adapted from the traditional backprop. Input vectors are passed through a feed-forward network to compute the cost function. $\mathcal{L}(\hat{y}^j, y^j)$ denotes the loss for the j-th task, to which a regularizer $\Omega(\theta^j_{inc}) = \Omega(\theta^j - \theta^{j-1}) = ||\theta^j - \theta^{j-1}||^2_{\text{fro}}$ is added to obtain the total cost $\mathcal{J}$, where θ contains all the weights (biases are considered as weights connected to the bias units). The increment of parameters rather than the parameters themselves are regularized, similar to the conceptor-aided SGD.

```
Require: Network depth, $l$
Require: $W^{(l)^j}, l \in \{1, \dots, L\}$, the weight matrices of the network
Require: $x^j$, one input vector of the j-th task
Require: $y^j$, the target output for $x^j$
    $h^{(0)} = x^j$
    for $l = 1, \dots L$ do
        $b^{(l)} = [h^{(l-1)\top}, 1]^\top$, include the bias unit
        $a^{(l)} = W^{(l)^j} b^{(l)}$
        $h^{(l)} = \sigma(a^{(l)})$
    end for
    $\hat{y}^j = h^{(l)}$
    $\mathcal{J} = \mathcal{L}(\hat{y}^j, y^j) + \lambda\Omega(\theta^j_{inc})$
```

Algorithm 2 The backward procedure of conceptor-aided backprop for the j-th task, adapted from the traditional backprop. The gradient g of the loss function $\mathcal{L}$ on the activations $a^{(l)}$ represents the error for the linear transformation $W^{(l)^j}$ between the $(l-1)$-th and the l−th layers. In the standard backprop algorithm, the gradient of $\mathcal{L}$ on $W^{(l)^j}$ is computed as an outer product of the post-synaptic errors g and the pre-synaptic activities $h^{(l-1)}$. This resembles the computation of the gradient in the linear SGD algorithm, which motivates us to apply conceptors in a similar fashion as in the conceptor-aided SGD. Specifically, we project the gradient $\nabla_{W^{(l)^j}}\mathcal{L}$ by the matrix conceptor $F^{(l-1)^j}$ that indicates the free memory space on the pre-synaptic layer.

```
    $g \leftarrow \nabla_{\hat{y}}\mathcal{J} = \nabla_{\hat{y}}\mathcal{L}(\hat{y}, y)$
for $l = L, L-1, \ldots, 1$ do
    Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation ($\odot$ denotes element-wise multiplication):
        $g \leftarrow \nabla_{a^{(l)}}\mathcal{J} = g \odot \sigma'(a^{(l)})$
    Compute the gradient of weights, project it by $F^{(l-1)^j}$, and add it to the regularization term on the increment:
        $\nabla_{W^{(l)^j}}\mathcal{J} = g(F^{(l-1)^j} b^{(l-1)})^\top + \lambda \nabla_{W^{(l)^j}}\Omega(\theta^j_{inc})$
            $= g b^{(l-1)^\top} F^{(l-1)^j} + 2\lambda W^{(l)\ j}_{inc}$
            $= g b^{(l-1)^\top} F^{(l-1)^j} + 2\lambda(W^{(l)^j} - W^{(l)^{j-1}})$
    Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
        $g \leftarrow \nabla_{h^{(l-1)}}\mathcal{J} = W^{(l)^{j\top}} g$
end for
```

2.4 experiments

We evaluate CAB on the permuted and disjoint MNIST tasks and compare it to *elastic weight consolidation* (EWC [Kirkpatrick et al., 2017]) and *incremental moment matching* (IMM [Lee et al., 2017]). EWC is a regularization-based method that uses the posterior distribution of parameters for the old tasks as a prior for the new task. They approximated the posterior by a Gaussian distribution with the parameters for old tasks as the mean and the inverse diagonal of the Fisher information matrix as the variance. IMM includes two variants which are mean-IMM and mode-IMM. Mean-IMM approximates the distribution of parameters for both old and new tasks by a Gaussian distribution, which is estimated by minimizing its KL-divergence from the mixture of two Gaussian posteriors, one for the old task and the other one for the new task. Mode-IMM estimates the mode of this mixture of two Gaussians and uses it as the optimal parameters for both tasks. Experimental results showed highly competitive performance of CAB.

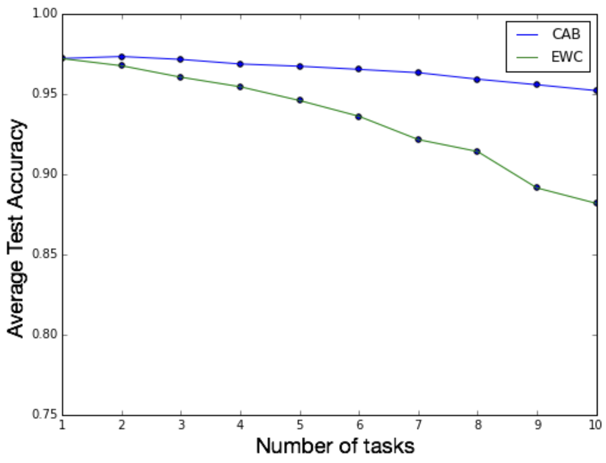
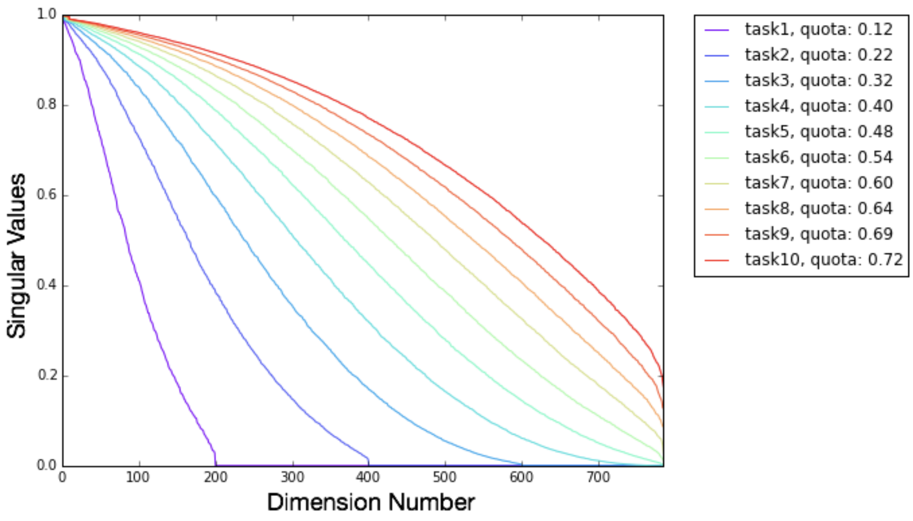


Figure 2.3: Average performance across already learned permuted MNIST tasks using CAB or EWC

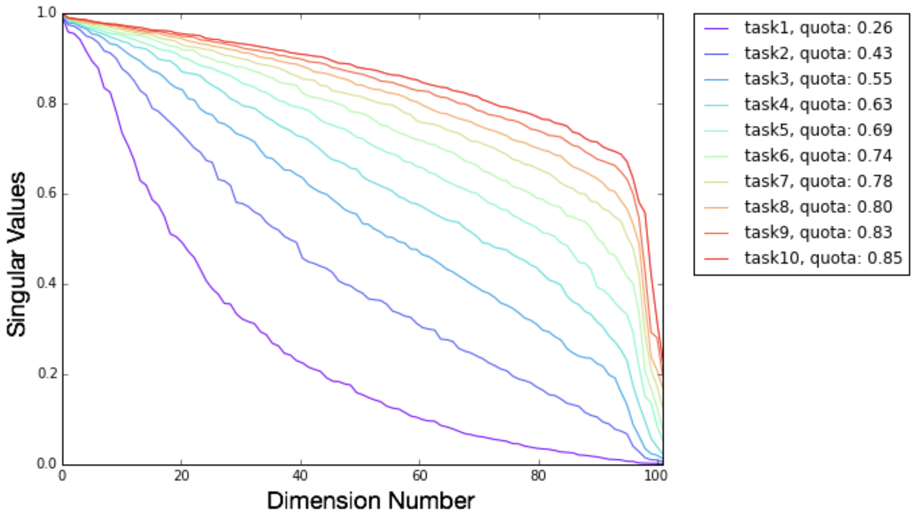
2.4.1 *Permuted MNIST Experiment*

In the permuted MNIST experiment [Goodfellow et al., 2014a; Kirkpatrick et al., 2017; Lee et al., 2017; Srivastava et al., 2013], a sequence of pattern recognition tasks are created from the MNIST dataset [LeCun, Cortes, and JC Burges, 1998]. For each task, a random permutation of input image pixels is generated and applied to all images in MNIST to obtain a new shuffled dataset, equally difficult for a MLP to recognize as the original one, the objective of each task is to recognize these images with shuffled pixels.

For a proof-of-concept demonstration, we trained a simple but sufficient feed-forward network with [784-100-10] of neurons to classify 10 permuted MNIST datasets. The network has logistic sigmoid neurons in both hidden and output layers, and is trained with mean squared error as the cost function. Vanilla SGD was used in all experiments to optimize the cost function. Learning rate and aperture were set to 0.1 and 4, respectively. For comparison, we also tested EWC on the same task with the same network architecture, based on the implementation by Seff [2017]. The parameters chosen for the EWC algorithm were 0.01 for the learning rate and 15 for the weight of the Fisher penalty term. Figure 2.3 shows the performance of CAB on this task, the average testing accuracy is 95.2% after learning all 10 tasks sequentially. Although a fair amount of effort was spent on searching for optimal parameters for EWC, the accuracies shown here might still not reflect its best performance. However, the same experiment with EWC was also conducted in [Kemker et al., 2017], where the authors reimplemented EWC on a network with higher capacity (2 hidden layers and 400 ReLU neurons per layer) and the resulting average accuracy after learning 10 tasks sequentially was shown to be around 93%.



(a) Singular value spectra of conceptors $A^{(0)^j}$ on the input layer.



(b) Singular value spectra of conceptors $A^{(1)^j}$ on the hidden layer.

Figure 2.4: The development of singular value spectra of conceptors for "used-up" space on the input layer and hidden layer during incremental learning of 10 permuted MNIST tasks. Quota of these conceptors are displayed in the legends.

Since all tasks are generated by permuting the same dataset, the portion of the input space occupied by each of them should have the same size. However, as more tasks are learned, the chance that the space of a new task will overlap with the already used input space increases. Figure 2.4 shows the singular value spectra and quota of the input and hidden layer conceptors every time after a new task is learned. As the incremental learning proceeds, it becomes less likely for a new task to be in the free space. For example, the second task increases the quota of the input layer memory space by 0.1, whereas the 10th task increases it by only 0.03. However, CAB still manages to make the network learn new tasks based on their input components in the non-overlapping space.

2.4.2 *Disjoint MNIST Experiment*

We then applied CAB to categorize the disjoint MNIST datasets into 10 classes [Lee et al., 2017; Srivastava et al., 2013]. In this experiment, the original MNIST dataset is divided into two disjoint datasets with the first one consisting of data for the first five digits (0 to 4), and the second one of the remaining five digits (5 to 9). This task requires a network to learn these two datasets one after the other, then examines its performance of classifying the entire MNIST testing images into 10 classes. The current state-of-the-art accuracy on this task, averaged over 10 learning trials, is $94.12(\pm 0.27)\%$, achieved by Lee et al. [2017] using IMM. They also tested EWC on the same task and the average accuracy was $52.72(\pm 1.36)\%$.

To test our method, we trained a feed-forward network with [784-800-10] neurons. Logistic sigmoid nonlinearities were used in both hidden and output layers, and the network was trained with vanilla SGD to minimize mean squared errors. The aperture $\alpha = 9$ was used for all conceptors on all layers, learning rate η and regularization coefficient λ were chosen to be 0.1 and 0.005 respectively. The accuracy of CAB on this task, measured by repeating the experiment 10 times, is $94.91(\pm 0.30)\%$. It is worth mentioning that the network used by Lee et al. [2017] for testing IMM and EWC had [784-800-800-10] rectified linear units (ReLU), so CAB achieved better performance with fewer layers and neurons.

2.4.3 *Computational Cost*

If a conceptor is computed by ridge regression, the time complexity is $O(nN^2 + N^3)$ when the design matrix is dense, where n is the number of samples and N the number of features. In terms of wall time measures, the time taken to compute a conceptor from the entire MNIST training set (in this case, $n = 55000$ images and $N = 784$ pixels, corresponding to the input layer in our networks) is 0.42 seconds of standard notebook CPU time on average. Although we did not implement it in these experiments, incremental online adaptation of conceptors by gradient descent is also possible in principle and would come at a cost of $O(N^2)$ per update.

2.5 DISCUSSION

The experiment results from the previous section indicate that learning disjoint MNIST datasets is a more challenging task for CAB than learning the permuted MNIST datasets. To see why this is the case, it is important to understand that when learning a new task, CAB exploits only the components of input vectors that are not inside the linear subspace characterized by the conceptor of previous tasks. In other words, the more overlap there is between the conceptor of the new task and that of the already learned tasks, the less components in the input patterns are left for CAB to correct the network's errors on the new task, hence more difficult for the network to learn these tasks sequentially.

To quantify the overlap between two conceptors C^i, C^j, we can use the similarity measure proposed by Jaeger [2014]:

$$\rho(C^i, C^j) = \frac{\|(S^i)^{1/2}(U^i)^\top U^j (S^j)^{1/2}\|_{\text{fro}}^2}{\|\text{diag}S^i\| \cdot \|\text{diag}S^j\|}, \tag{2.21}$$

where $C^i = U^i S^i (U^i)^\top$ and $C^j = U^j S^j (U^j)^\top$ are their singular value decompositions. This measure ranges in $[0, 1]$. It is 0 if and only if C^i, C^j specify two orthogonal linear subspaces, and 1 if and only if C^i is a multiple of C^j.

In order to compare the difficulties of the permuted and disjoints MNIST experiments, we selected four datasets: $D_{original}$, $D_{permuted}$, D_{5to9} and D_{0to4}, where $D_{original}$ is the original MNIST dataset; $D_{permuted}$ is the whole MNIST dataset but the pixels of every image is shuffled by the same randomly generated permutation; D_{5to9} consists of only the images of digits 5 to 9 from the MNIST dataset, and D_{0to4} has only the images of digits 0 to 4. Then for each of these four datasets, we computed a conceptor from the raw input images data inside it. The results were four conceptors $C_{original}, C_{permuted}, C_{5to9}$ and C_{0to4}.

In the permuted MNIST experiment, the network has to learn to recognize $D_{permuted}$ and $D_{original}$ sequentially, the overlap between their corresponding conceptors can be measured by $\rho(C_{original}, C_{permuted})$, which is around 0.3 on average.

In contrast, $\rho(C_{5to9}, C_{0to4})$ is much higher (≈ 0.95), which means the input images in D_{5to9} and D_{0to4} span roughly the same linear subspaces of the input memory space. Therefore, if a network is first trained on D_{5to9} and then on D_{0to4}, only a very small amount of components of the images in D_{0to4} can be exploited to compensate the errors, namely those components preserved by $F_{5to9} := \neg C_{5to9}$, whereas the linear transformation of the components inside the subspace characterized by C_{5to9} will be fixed. Figure 2.5 columns (a) and (b) show some images from D_{0to4} projected by C_{5to9} and F_{5to9}. Note that when learning to recognize the second dataset, CAB only allows the network to adjust its performance based on the projected versions displayed in column (b), which are much less legible than the images projected by C_{5to9}, shown in the column (a).

Since the same images are also included in $D_{original}$, for comparison, we also visualized their components inside the linear subspaces characterized by $C_{permuted}$ and $F_{permuted}$, which can be found in Figure 2.5 columns (c) and (d). In the setting of permuted MNIST experiment, after the network is trained on $D_{permuted}$, it can only rely on the components displayed in column (d) to compensate its output errors. However, it is clear that the images in column (d) are more distinguishable than those in column (b), hence the permuted MNIST experiment is easier for CAB than the disjoint one.

A direction for improvement of CAB, suggested by the analysis above, is to change CAB such that the weights of the network can be adjusted even when the input patterns of different tasks lie in the same linear subspace. On the other hand, such similarity between tasks might be exploited to save training time. So another question for further investigation is how to turn the similarity between different tasks into a desirable property rather than difficulty for continual learning. In the next chapter, we propose one alternative solution based on conceptors to address these problems.

2.6 CONCLUSION

In this chapter, we proposed a conceptor-aided backprop algorithm by applying a conceptor to every linear layer of a feed-forward network. This method uses conceptors to guide gradients of parameters during the back-propagation procedure. As a result, learning a new task interferes only minimally with previously learned tasks, and the amount of already used network capacity can be monitored via the singular value spectra and quota of conceptors.

In [Jaeger, 2014], different scenarios for continual learning are investigated in a reservoir computing setting. Two extreme cases are obtained when (i) the involved learning tasks are entirely unrelated to each other, versus (ii) all tasks come from the same parametric family of learning tasks. The two cases differ conspicuously with regards to the geometry of involved conceptors, and with regards to opportunities to re-use previously acquired functionality in subsequent learning episodes. The permuted MNIST task is an example of (i) while the disjoint MNIST task rather is of type (ii). Conceptors provide an analytical tool to discuss the "family relatedness" and enabling/disabling conditions for continual learning in geometrical terms.

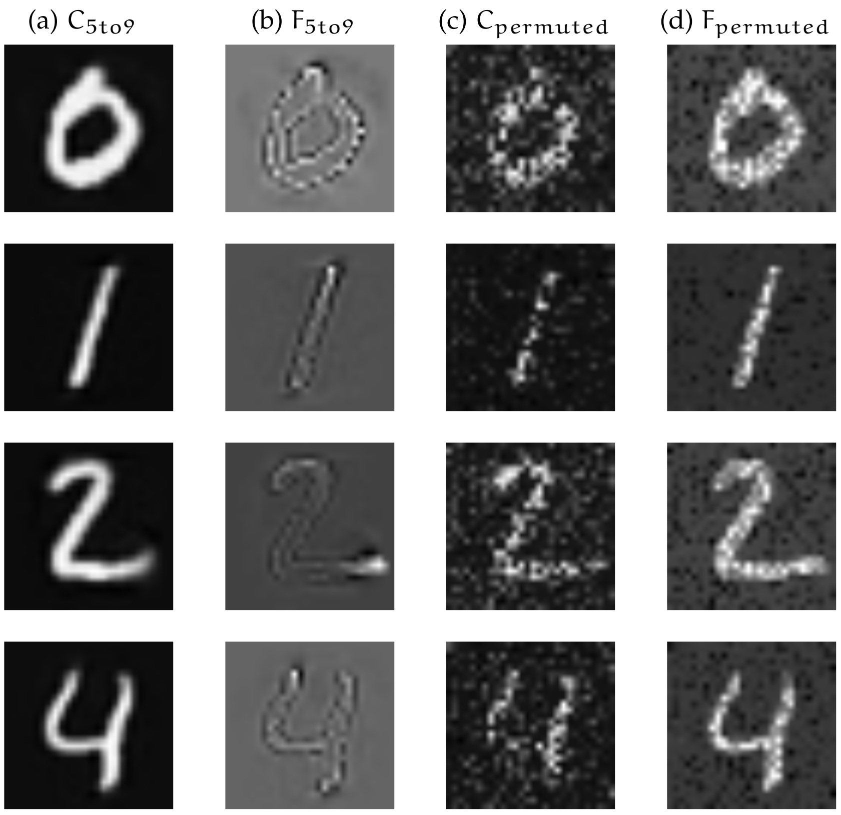


Figure 2.5: Projecting several MNIST images for digits 0 to 4 by different conceptors. Columns (a) and (b) are results after projection by Conceptors C_{5to9} and its negation F_{5to9}. After a network is trained on digits 5 to 9, CAB will only use the components projected by F_{5to9} to correct the classification errors on these images. Columns (c) and (d) are results after projection by the conceptor $C_{permuted}$ computed from the shuffled MNIST dataset and its negation $F_{permuted}$. After a network is trained on the shuffled MNIST, CAB will only use the components projected by $F_{permuted}$ to correct the classification errors on these images.

3 CONCEPTOR-BASED PSEUDO-REHEARSAL

We have seen that conceptor-aided backpropagation (CAB) can alleviate catastrophic forgetting in a feed-forward network. CAB uses conceptors to characterize the linear subspaces of the input space already used by representations of previous tasks. During the backpropagation procedure for learning new tasks, CAB exploits only the components outside the already occupied subspace, thus learning new tasks does not interfere with the old tasks.

For many machine learning systems, however, different tasks may share the same input space but require distinct outputs. For example, an agent that needs to solve several vision-based tasks always receives natural images as input. Therefore, the input distributions for all these tasks are exactly the distribution of natural images. In this case, training such agent using CAB is difficult, since the input vectors of new tasks are completely inside the space occupied by previous tasks, and almost no input components are available for further adaptation.

In this chapter, to overcome this limitation of CAB, we first study the relationship between conceptors and weight matrices trained using L2 regularization, which motivates us to propose another conceptor-based continual learning method based on pseudo-rehearsal. In particular, we modify the original approach of CAB so that instead of shielding the weight matrices based on the subspace in the pre-synaptic layer, the *Conceptor-based Pseudo-Rehearsal* (CPR) protects them using the subspace of the post-synaptic layer. As a result, new activation components triggered by the change of parameters appears only in the free space, thus it does not interfere with old tasks. Moreover, helpful features from the old tasks can be used by the new tasks, which allows a forward transfer learning in the network [Lopez-Paz et al., 2017].

To test CPR empirically and demonstrate that it can overcome the drawback of CAB, we designed an incremental classifier learning experiment, in which a multi-class classification task is divided into a sequence of One-vs-All [Rifkin and Klautau, 2004] binary classification tasks. Experiment results show that, with the proposed method, a multilayer perceptron (MLP) is able to incrementally learn the binary classification subtasks and achieve a final multi-label accuracy that matches the performance of a jointly trained network. The rest of this chapter is a verbatim copy of He [2018a] with modifications to make it congruent with the previous chapter.

3.1 CONCEPTOR AND L2 REGULARIZATION

In this section, we discuss the relationship between conceptors and L2 regularization, which will later help understand how the proposed method works.

3.1.1 *Relationship to Ridge Regression*

Consider a training dataset $D = \{(x_1, y_1), \ldots (x_n, y_n)\}$ where $x_i \in \mathbb{R}^N, y_i \in \mathbb{R}^M$. A linear map W that fits the dataset D by minimizing $\mathcal{J}_{linear}(W) := \sum_{i=1}^{n} \|y_i - Wx_i\|^2$ is given by linear regression:

$$W_{linear} := YX^\top (XX^\top)^{-1}, \tag{3.1}$$

where X is a $N \times n$ data collection matrix whose i-th column is x_i and Y is the $M \times n$ output data collection matrix whose i-th column is y_i.

If we add a L2 regularization term, and minimize $\mathcal{J}_{ridge}(W, \alpha) := \sum_{i=1}^{n} \|y_i - Wx_i\|^2] + \alpha^{-2}\|W\|_{\text{fro}}^2$ instead, we obtain the ridge regression solution:

$$W_{ridge} = YX^\top (XX^\top + \alpha^{-2} I)^{-1}. \tag{3.2}$$

Recall that a conceptor matrix trained on the input vectors in D is a regularized identity map that minimizes $\mathcal{J}(C, \alpha) := \sum_{i=1}^{n} \|x_i - Cx_i\|^2] + \alpha^{-2}\|C\|_{\text{fro}}^2$ and the solution can be computed in closed form as

$$C = XX^\top (XX^\top + \alpha^{-2} I)^{-1}. \tag{3.3}$$

Combine 3.1, 3.2 and 3.3, it is easy to see that

$$W_{ridge} = W_{linear}(XX^\top)(XX^\top + \alpha^{-2} I)^{-1} = W_{linear} C. \tag{3.4}$$

Hence, ridge regression is a composition of conceptor projection and linear regression, and it follows that

$$\forall x \in \mathbb{R}^N, Cx = 0 \implies W_{ridge} x = 0. \tag{3.5}$$

Suppose the singular value decomposition (SVD) of X has the form $X = UDV^\top$ with U being an $N \times N$ orthogonal matrix and V being an $n \times N$ matrix with orthonormal columns. D is a diagonal matrix with descending diagonal entries $\sigma_i \geqslant 0$. The SVD of C can be written in terms of the SVD of X:

$$C = U \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2+\alpha^{-2}} & & \\ & \ddots & \\ & & \frac{\sigma_N^2}{\sigma_N^2+\alpha^{-2}} \end{bmatrix} U^\top. \tag{3.6}$$

Since α^{-2} and σ_i^{-2} are non-negative, the singular values of C are between 0 and 1. Futhermore, $\frac{\sigma_i^2}{\sigma_i^2+\alpha^{-2}} \approx 0$ for $\sigma_i^2 \ll \alpha^{-2}$. Therefore, the vectors x that satisfy $Cx \approx 0$ are the vectors that live in the linear subspace spanned by principal components (columns of U) with small variance d_i^2. Given the implication of 3.5, we know that these vectors will also be ignored by the ridge weights $W_{\mathtt{ridge}}$. This is sometimes called the shrinkage effect [Hastie, Tibshirani, and Friedman, 2001] of ridge regression.

3.1.2 *Relationship to Weight Decay*

One might wonder if a similar shrinkage effect also exists for weight matrices connecting two layers of neurons in a deep network when they are trained with L2 regularization (or weight decay [Krogh and Hertz, 1992]). Unfortunately this cannot be shown analytically, but we can check it empirically. The implication $\forall x \in \mathbb{R}^N, Cx \approx 0 \implies W_{L2}x \approx 0$ holds if W_{L2} nulls vectors in the linear subspace spanned by principal components (PCs) corresponding to very low variances. In other words, if we compute the matrix $\tilde{W}_{L2}$ that represents the same linear transformation as W_{L2} but changes the basis of input space to the PCs: $\tilde{W}_{L2} := W_{L2}U$, the norms of the last columns in $\tilde{W}_{L2}$ should be close to 0.

To verify our hypothesis that L2 regularization will result in similar shrinkage weights, we trained two feed-forward networks with $[784 - 800 - 800 - 10]$ neurons on the MNIST classification task, one with L2 regularization and the other one without. In this experiment, we only consider the weight matrices that connect the two layers of hidden neurons. We denote the weight matrix trained with L2 regularization as W_{L2}, and the one without as W. After training the network, we computed the principal components of neuron activation in the first hidden layer of each network and collected these PCs as columns of U and U_{L2}, respectively. Figure 3.1 visualizes the norms of columns in $\tilde{W} = WU$ and $\tilde{W}_{L2} = W_{L2}U$. It can be seen that with L2 regularization, the columns correspond to PC directions with low variances have almost zero norms, and this agrees with the nulling effect of conceptors.

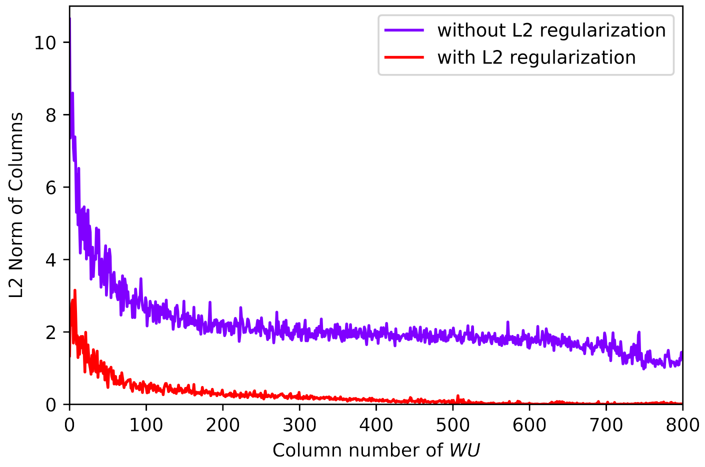


Figure 3.1: The L2 norm of columns in WU. When W is trained with L2 regularization, the norm of the last columns are close to 0, hence vectors in the linear subspace spanned by principal components corresponding to small variances will be nulled by W. This shrinkage effect coincides with the nulling effect of conceptors.

3.2 LIMITATIONS OF CAB

In this section, we discuss in detail the limitation of CAB when the input space of a new task overlaps with that of the previous tasks. For simplicity, we illustrate the idea by studying the scenario of sequentially training two supervised tasks on a 3-layer MLP.

Let $T^1 = \{(x_i^1, y_i^1)\}_{i \in I^1}, T^2 = \{(x_i^2, y_i^2)\}_{i \in I^2}$ be two tasks to be trained on a MLP with $L = 3$ layers: $\forall l \in \{0, \dots, L-1\}$

$$a_l = [\hat{a}_l^\top, 1]^\top, \tag{3.7}$$

$$\hat{a}_{l+1} = f_l(W_l a_{l-1}), \tag{3.8}$$

where W_l, f_l are weights and element-wise nonlinear transfer functions (e.g. tanh, ReLU [Nair and Hinton, 2010]) for layer l. $\hat{a}_0 = x$ is the input vector. CAB trains the network on the first task by minimizing the objective function

$$\mathcal{J}^1 = \mathcal{L}(\hat{a}_L(x^1, \{W_l\}_{l \in \{0,\dots,L-1\}}), y^1) + \lambda \sum_{l=0}^{L-1} \|W_l\|_{\mathrm{fro}}^2, \tag{3.9}$$

here λ is a regularizer parameter and $\mathcal{L}$ is some loss function depending on the network output $\hat{a}_L$ and the target y^1 of task 1. After the network is trained on T^1, the resulting weights that minimize the loss are saved as $\{W_l^1\}_{l\in\{0,\dots,L-1\}}$ and a batch of input vectors from $\{x_i^1\}_{i\in I^1}$ are fed into the trained network to collect a set of activation vectors $\{a_{l\,i}^1\}$ from each layer l. From these activation vectors, a conceptor C_l^1 is computed to characterize the linear subspace S_{C_l} already used by the first task in layer l. Importantly, C_l^1 preserves the activation vectors when the inputs are from T^1, and its negation $F_l^1 := \neg C_l^1 = I - C_l^1$ that characterizes the orthogonal complement of S_{C_l} (which is considered the free space S_{F_l}) will null these vectors:

$$C_l^1 a_{l\,i}^1 \approx a_{l\,i}^1, \tag{3.10}$$

$$F_l^1 a_{l\,i}^1 \approx 0. \tag{3.11}$$

When the second task comes, the new weights W_l^2 (which will later replace W_l^1) are computed by adding incremental weights W_l^{inc} to the old weights $W_l^1 : W_l^2 := W_l^1 + W_l^{\text{inc}}$. In the training process of the second task, W_l^1 will be fixed and only W_l^{inc} will be adapted to minimize

$$\mathcal{J}^2 = \mathcal{L}(\hat{a}_L(x^2, \{W_l^2\}_{l\in\{0,\dots,L-1\}}), y^2) + \lambda \sum_{l=0}^{L-1} \|W_l^{\text{inc}}\|_{\text{fro}}^2, \tag{3.12}$$

$$\text{subject to } W_l^{\text{inc}} C_l^1 \approx 0. \tag{3.13}$$

CAB enforces the constraint in 3.13 by multiplying the gradient of W_l^{inc} with F_l^1 at every step of the gradient descent, which essentially projects the rows of W_l^{inc} to the orthogonal complement of S_{C_l}, thus $W_l^{\text{inc}} C_l^1 \approx 0$. As a result, the final weight W_l^2 does not forget the first task:

$$\begin{aligned} \forall l \in \{0,\dots,L-1\}, f_l(W_l^2 a_l^1) &= f_l((W_l^1 + W_l^{\text{inc}})a_l^1) \\ &\approx f_l(W_l^1 a_l^1 + 0) = f_l(W_l^1 a_l^1). \end{aligned} \tag{3.14}$$

Although effective at preventing forgetting, the problem with CAB is that it might not be able to learn new tasks when their input spaces overlap significantly with the old tasks. Consider an extreme case where two tasks have exactly the same inputs but different outputs: $\{x_i^1\}_{i\in I^1} = \{x_i^2\}_{i\in I^2}, \{y_i^1\}_{i\in I^1} \neq \{y_i^2\}_{i\in I^2}$ (for example, $\{x_i^1\}_{i\in I^1}$ is the entire set of MNIST digit images, y_i^1, y_i^2 are binary classification outputs such that $y_i^1 = 1$ only if x_i^1 is an image of digit 0 and $y_i^2 = 1$ only if x_i^2 is an image of digit 1). In this case, it is possible to train both tasks on the same network with a multi-head output (i.e., using a separate matrix W_{L-1}^j at the last layer for each task j), which is commonly used in continual learning models (Bakker and Heskes [2003], Li and Hoiem [2016], Rusu et al. [2016], and Serra et al. [2018]). However, CAB will perform poorly in this case because enforcing $W_l^{\text{inc}} C_l^1 \approx 0$ will render W_l^{inc} useless. This can be shown by induction:

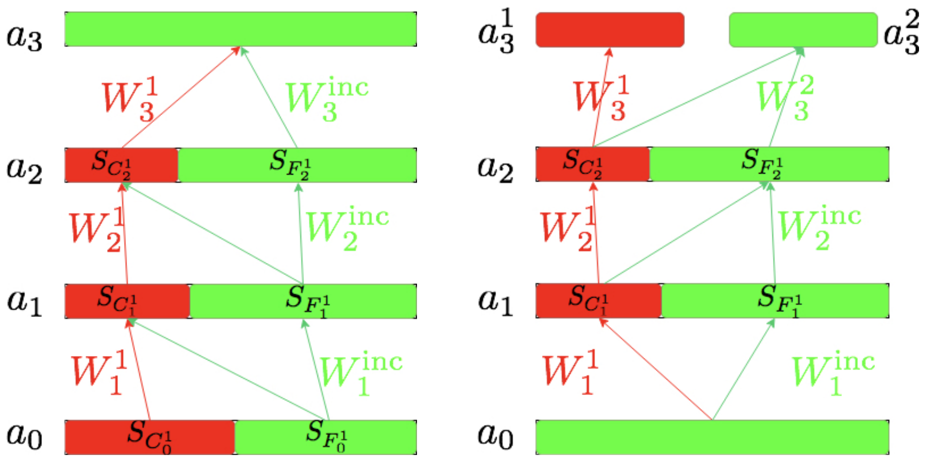
since the input vectors of both tasks are the same, $a_0^2 = a_0^1 \in S_{C_0^1}$, $\forall l \in \{0, \dots, L-1\}$, if $a_l^2 = a_l^1$, then

$$\begin{aligned} a_{l+1}^2 &= [f(W_l^1 a_l^2 + W_l^{\text{inc}} a_l^2)^\top, 1]^\top \approx [f(W_l^1 a_l^1 + W_l^{\text{inc}} C_l^1 a_l^1)^\top, 1]^\top \\ &\approx [f(W_l^1 a_l^1)^\top, 1]^\top = a_{l+1}^1. \end{aligned} \tag{3.15}$$

Therefore, the output of the network will remain the same and it cannot adapt to the second task.

3.3 CONCEPTOR-BASED PSEUDO REHEARSAL

In this section, we introduce the conceptor-based pseudo rehearsal (CPR), which does not have the drawback discussed in the previous section. Since the intransigence of CAB is caused by task overlap in the input space, we usually have no control over representations in the input space because they are given by the data source. Instead of constraining the incremental weights by the free space of the input layer, the new method constrains them by the free space of the output layer. Figure 3.2 visualizes this difference between CPR and CAB.



(a) CAB (b) CPR

Figure 3.2: Structure of a network trained on two tasks incrementally using CAB and CPR from layer a0 to a3. The red region in each layer represents the subspace already occupied by the first task, and the green region represents the free space available to the second task. By constraining the incremental weights W_i^{inc} based on the free space in post-synaptic layers instead of pre-synaptic layers, CPR makes the entire input layer available to all tasks, thus can overcome the drawback of CAB. By tracing the red arrows that represent old weights from the input layer a_0 to the output layer a_3, one can verify that both methods avoid catastrophic forgetting.

Again, we illustrate the idea by considering the scenario of training two tasks $T^1 = \{(x_i^1, y_i^1)\}_{i \in I^1}, T^2 = \{(x_i^2, y_i^2)\}_{i \in I^2}$ sequentially on a feed-forward network. A complete algorithm that extends to any finite number of layers and tasks is given at the end of the section.

To learn the first task, the procedure remains the same as with CAB. For the second task, for $l \neq 3$, the new weight $W_l^2 := W_l^1 + W_l^{\text{inc}}$ is defined as the sum of the weight W_l^1 learned from the previous task and an incremental weight W_l^{inc}; for $l = 3$, we use a multi-head output, so W_l^2 is a new weight matrix independent from W_l^1 (see Figure 3.2 (b)). During the training process for T^2, W_l^1 will be considered as a constant and only W_l^{inc} will be adapted. Unlike CAB, to ensure that adding W_l^{inc} does not interfere with the previous task, we replace the constraint $W_l^{\text{inc}} C_l^1 \approx 0$ in 3.13 by

$$\forall a_l, C_{l+1}^1 f_l(W_l^2 a_l) \approx C_{l+1}^1 f_l(W_l^1 a_l). \tag{3.16}$$

The intuition behind this constraint is that changing the weights from W_l^1 to W_l^2 should not result in any change of the activation component in the already used subspace

$S_{C_{l+1}}$ of the next layer. This forces the change caused by the incremental weight W_l^{inc} to only happen in the free space $S_{F_{l+1}}$ of the next layer. Note that CAB constrains the incremental weights W_l^{inc} of layer l by the conceptor C_l^1 in the pre-synaptic layer, whereas CPR constrains W_l^{inc} by the conceptor C_{l+1}^1 in the post-synaptic layer.

In practice, the constraint 3.16 can be enforced by a pseudo-rehearsal strategy, which adds the following rehearsal loss to the objective function of the second task described in 3.12:

$$\gamma \sum_{l=0}^{L-2} \mathbb{E}[\|C_{l+1}^1(f_l(W_l^2\epsilon_l) - f_l(W_l^1\epsilon_l))\|_2^2]. \tag{3.17}$$

Here ϵ_l is a random vector that has the same shape as a_l and we simply draw its samples from the standard normal distribution, these random samples can be considered as the pseudo input vectors for rehearsing the weights in layer l, and its corresponding output vectors given by the old weights in the used space $C_{l+1}^1 f_l(W_l^1\epsilon_l)$ are the pseudo targets. We use standard normal distribution instead of a learned distribution of activation vectors a_l^1 from the previous task because we need the constraint $C_{l+1}^1 f_l(W_l^2 a_l) \approx C_{l+1}^1 f_l(W_l^1 a_l)$ to hold for any vector a_l, which might be samples of a different distribution. This is a necessary condition for the proof in 3.20.

In Section 3.1, we discussed the relationship between conceptors and the weights trained with L2 regularization, and we concluded with the implication in 3.5. We can now use this implication to show that when the constraint 3.13 is satisfied, the change of weights does not lead to forgetting of previous tasks.

Formally, for any activation vector a_l, if

$$C_{l+1}^1(f_l(W_l^2 a_l) - f_l(W_l^1 a_l)) \approx 0, \tag{3.18}$$

by 3.5, it implies that

$$W_{l+1}^1(f_l(W_l^2 a_l) - f_l(W_l^1 a_l)) \approx 0. \tag{3.19}$$

It can be shown by induction that with the new parameters $\{W_l^2\}_{l\in\{0,1\}}$ and the task-specific output weight W_2^1, the MLP does not forget the first task:

$$\begin{aligned}
&f_2(W_2^1 f_1(W_1^2 f_0(W_0^2 x_i^1))) \\
=&f_2(W_2^1[f_1(W_1^2 f_0(W_0^2 x_i^1)) - f_1(W_1^1 f_0(W_0^2 x_i^1)) + f_1(W_1^1 f_0(W_0^2 x_i^1))]) \\
\approx&f_2(W_2^1 f_1(W_1^1 f_0(W_0^2 x_i^1))) \\
=&f_2(W_2^1 f_1(W_1^1[f_0(W_0^2 x_i^1) - f_0(W_0^1 x_i^1) + f_0(W_0^1 x_i^1)])) \\
\approx&f_2(W_2^1 f_1(W_1^1 f_0(W_0^1 x_i^1))).
\end{aligned} \tag{3.20}$$

On the other hand, as long as the old task does not occupy the entire space of the $(l+1)$-th hidden layer (in other words, C^1_{l+1} is not almost the identity matrix), the regularization term does not imply $f_l(W^2_l a_l) = f_l(W^1_l a_l)$, thus W^{inc}_l can still be adapted. A way to check how much capacity is left for further learning is, as we described before, to plot the singular value spectra of C^1_{l+1}, if there are still some singular values close to 0, $S_{C^1_{l+1}}$ does not equal to the entire space and the network is capable of learning more. Another advantage of CPR over CAB is that conceptors are applied to the hidden layers instead of the input layers in CPR. In the case that the entire space of a hidden layer is occupied, the capacity can be extended by simply introducing new neurons in the overloaded hidden layer when using CPR. This is normally not possible with CAB since the size of the input layer is usually decided by the data and has to stay fixed.

Below is the complete version of the CPR algorithm that trains a network with L layers on a sequence of J supervised tasks using CPR ($\forall j \in \{1,\dots,J\}, T^j = \{(x^j_i, y^j_i)\}_{i \in I^j}$) :

- **Initialization (no task trained yet):**
  $\forall l = 1,\dots,L-1$, set A^0_l to zero. $\forall l = 0,\dots,L-1$, set W^0_l to zero and randomly initialize W^{inc}_l.

- **Incremental task learning:** For $j = 1,\dots,J$ do:
  1. Let $W^j_l := W^{j-1}_l + W^{\text{inc}}_l$ for $l < L$ and $W^j_L := W^{\text{inc}}_L$. Update W^{inc}_l by stochastic gradient descent to minimize $\mathcal{J} = \mathcal{L}(a_L(x^j, \{W^j_l\}_{l \in \{0,\dots,L-1\}}), y^j) + \lambda \sum_{l=0}^{L-1} \|W^{\text{inc}}_l\|^2_{\text{fro}} + \gamma \sum_{l=0}^{L-2} \mathbb{E}[\|C^1_{l+1}(f_l(W^2_l \epsilon_l) - f_l(W^1_l \epsilon_l))\|^2_2]$. The gradient of $\mathcal{J}$ with respect to W^{inc}_l is estimated using a batch of samples for x^j, y^j and ϵ_l for $l = 0,\dots,L-2$. The samples for ϵ_l are randomly drawn from the standard normal distribution.
  2. After training on the j-th task, run the forward procedure again on a batch of input vectors from the j-th training dataset, and collect activations $\{a^j_{li}\}$ to compute a conceptor C^j_l.
  3. Update the conceptor for already used space using the OR operation: $\forall l = 1,\dots,L-1, A^j_l = A^{j-1}_l \vee C^j_l$

3.4 EXPERIMENTS

10 BINARY MNIST To clearly demonstrate that the proposed method can overcome the limitation of CAB discussed in Section 3.2, we designed an experiment where a sequence of tasks to be learned have exactly the same input space, but the output of every task varies. In particular, we turn the standard task of classifying MNIST digits into a sequence of 10 binary classification subtasks, where every task is trained on

the images from the entire MNIST training set, but for a different task the network has to detect a different digit class by returning 1 for images from that class and 0 otherwise. For example, it should return 1 for the first task only if the input is an image of digit 0. After the 10 binary classification tasks are finished, we test the network by its performance on classifying the testing images into 10 classes.

The network architecture we use for this experiment has $[784 - 800 - 800 - 10]$ neurons. We use rectified linear units for the hidden neurons, and the sigmoid transfer function for the output neurons. For one binary classification subtask, only a single output neuron is trained and used. Weight matrices in all layers and for all tasks are randomly initialized by Xavier initializer [Glorot and Bengio, 2010]. The loss function is optimized by the Adam optimizer [Kingma and Ba, 2014] with default parameters provided by Tensorflow [Abadi et al., 2015]. Table 3.1 compares the final accuracies on the entire MNIST testing dataset using different methods.

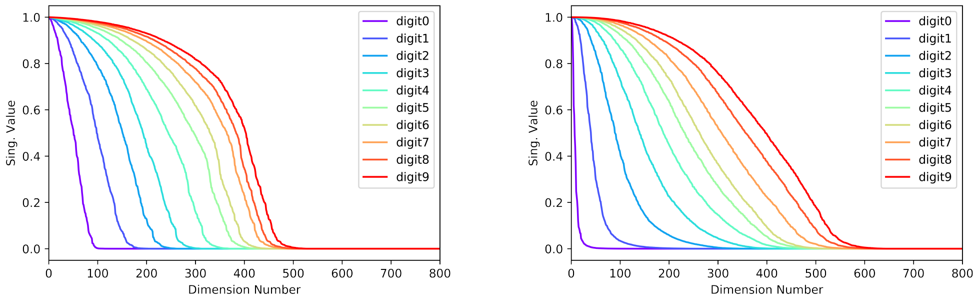
We first trained a standard classifier with the above-mentioned architecture as a control experiment. We call this scheme "Joint Training", since the 10 classification tasks are trained at the same time. An ideal incremental learning method should achieve an accuracy close to this case.

For CAB, if we choose a large aperture $\alpha = 6$, the binary classification accuracy decreases to almost chance level after the first 3 subtasks, since the gradients are nulled by conceptor projection and new subtasks will not be learned. If we adopt a very small aperture $\alpha = 10^{-7}$, CAB is equivalent to normal back-propagation, and the network learns every subtask perfectly but will also forget catastrophically. Among all apertures we tried, the best performance achieved is when $\alpha = 0.15$, in which case the conceptors partially protect past knowledge, and achieve an optimal balance between learning and forgetting. However, due to their shared input space, the subtasks are intrinsically conflicting with each other from the perspective of CAB. Therefore, the overall accuracy is still far from the joint training scheme.

With aperture $\alpha = 4$ for all conceptors, weight decay parameter $\lambda = 9 \times 10^{-6}$ and pseudo-rehearsal strength $\gamma = 10$, CPR could achieve an accuracy almost as good as that of the joint training scheme. Since the last layer of the network is a multi-head operation, every task has its own output weights. One might argue that the performance of CPR can be achieved by freezing the first two layers and only adapting the output weights, which is a common transfer learning technique [Yosinski et al., 2014]. For this reason, we also conducted an experiment where only the last layer is adapted after the first subtask is trained. However, this method also performs poorly since the hidden features for detecting digit 0 is not adequate for classifying all 10 digits. CPR, on the other hand, allows new features to be developed in the hidden layers during subsequent training. Figure 3.3 shows how the singular values of conceptors on the two hidden layers grow as more tasks are learned.

Table 3.1: Accuracies on 10 Binary MNIST Task using different methods

Method	Testing Accuracy
Joint Training	98.18% (best achieved in five runs)
CAB ($\alpha = 6$)	23.74%
CAB ($\alpha = 10^{-7}$)	49.16%
CAB ($\alpha = 0.15$)	69.73% (best achieved in five runs)
Only Last Layer	64.59%
CPR ($\alpha = 4$)	98.032($\pm$0.02)% (averaged over five runs)



(a) hidden layer 1 (b) hidden layer 2

Figure 3.3: Developments of singular value spectra of conceptors for already occupied space in two hidden layers as more binary classification tasks are learned.

SPLIT MNIST Another experiment we conducted is to split the original MNIST dataset into two disjoint subsets each contains images of five digits ($0-4$ in the first dataset and $5-9$ in the second) and compute the average accuracy after learning them one after the other. In this experiment, the input spaces of two tasks overlap with each other but they are not exactly the same. The best result on this experiment was 99.0% achieved by [Serra et al., 2018] after 50 epochs of training. With CPR, the same network architecture can achieve a final accuracy of 99.05% after 10 epochs. The hyper-parameters adopted are exactly the same as those used in 10 Binary MNIST experiment.

PERMUTED MNIST Finally, we tested CPR on the permuted MNIST experiment, in which we randomly shuffle the pixels in MNIST to create another dataset of the same size as the original MNIST, and compute the average performance after learning them one after the other. In this setup, due to the random permutation, the input spaces between two tasks intersect minimally. CAB was shown [He and Jaeger, 2018] to outperform two other popular approaches [Kirkpatrick et al., 2017; Lee et al., 2017] to continual learning on this task. We tested both methods on the network architecture described in the first experiment. CAB achieves a final average accuracy of 97.34% with $\alpha = 4$. CPR performs almost as good as CAB with a final accuracy at 97.3% using the same aperture and other hyper-parameters as in the 10 Binary MNIST experiment.

3.5 CONCLUSION

In this chapter, we discussed the limitation of conceptor-aided backprop on incrementally training tasks with similar input spaces. To overcome this limitation, we proposed another continual learning method called conceptor-based pseudo-rehearsal. In contrast to CAB, CPR constrains subsequent changes of a weight matrix based on the subspace in post-synaptic layer instead of pre-synaptic layer.

To understand how and why CPR works, we also discussed the relationship between conceptors and weights trained under L2 regularization. Conventionally, L2 regularization has been used to prevent a model from learning noise in training data in order to achieve better generalization. CAB and CPR show that the capacity saved from learning randomness in one dataset can be identified by conceptors and utilized for learning new tasks.

4 TASK AGNOSTIC CONTINUAL LEARNING VIA META LEARNING

Most continual learning approaches we have reviewed before implicitly assume that there exists a multi-task solution for the sequence of tasks and that task information is available to the learner. In this chapter, we discuss realistic scenarios where these assumptions do not hold. A natural approach to deal with this case is to separate the concerns into *what* task is currently being solved and *how* the task should be solved. This approach allows us to move the focus of continual learning from less forgetting to faster remembering – i.e measuring how quickly the network recovers performance rather than measuring the network's performance without any adaptation. It also opens the door to combining meta-learning and continual learning techniques, leveraging their complementary advantages. In practice, this *What and How* framework is implemented by differentiating task specific parameters from task agnostic (meta) parameters, where the latter are optimized in a continual meta learning fashion, without access to multiple tasks at the same time. The rest of this chapter is an almost verbatim copy of [He et al., 2020], the main idea of which was first introduced in [He et al., 2019b].

4.1 INTRODUCTION

In the previous two chapters we have introduced two conceptor-based continual learning (CL) algorithms. Like many other CL algorithms, their goal is to ensure that, after training on a sequence of tasks, the performance of the network is close to a network trained on all tasks at the same time. Hence, all these methods implicitly assume that there is always a multi-task solution that fits all previous tasks. In addition, they are designed with an assumption that either the identities of different tasks or the boundaries between them are available to the continual learner so that it knows when to update the conceptors to protect the already used linear subspace. Such an assumption is also commonly made by many CL methods [Zeno et al., 2018] in order to decide what parameters to consolidate and at what time.

However, there are many scenarios where both of the above-mentioned assumptions do not hold. Consider a multi-agent game in reinforcement learning (RL), where all agents are learning and adapting their policies. For any of these agents, the objective it tries to optimize (in other words, its task) depends not only on itself and the environment, but also on the policies and configurations of other agents, which are usually not directly observable. Moreover, the other agents might change their policies at any moment as they are also learning. As a result, the task for this agent is changing all the time and there are no clearly defined boundaries available to the agent. It has been observed that such non-stationarity in multi-agent systems usually causes catastrophic forgetting of the agent [Hernandez-Leal, Kartal, and Taylor, 2019]. For example, Vinyals et al. [2019] trained agents to play the video game StarCraft II by self-play [Tesauro et al., 1995], and they noticed that one salient drawback of this approach is in fact forgetting: the agent may forget how to defeat a previous version of itself as training progresses, and this may lead to a "tail-chasing" cycle where the agents always relearn a previously learned strategy and training never converges.

Furthermore, since the tasks now depend on the configurations of other agents, in general, there is no guarantee that a multi-task solution would exist in these settings. A potential example is Generative Adversarial Networks (GANs) [Goodfellow et al., 2014b], where a generator G and a discriminator D are trained together by playing a minimax game. The objective of D is to classify the data as real or fake, whereas the goal of G is to fool D as much as possible by generating fake data. It was shown in [Goodfellow et al., 2014b] that the optimal discriminator $D^*(x)$ is a function of the generator probability density function $p_G(x)$,

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}.$$

Therefore, if we take two snapshots G_1, G_2 of the generator at different moments of the training process such that $p_{G_1}(x) \neq p_{G_2}(x)$ for some x where $p_{data}(x) \neq 0$, then their corresponding optimal discriminators have to be different. In other words, there is no multi-task solution for the discriminator to be optimal for both generators. As a result, optimizing the discriminator with respect to a new version of the generator will inevitably lead to degradation of its performance with respect to a past generator, which is considered *forgetting* by the traditional metric of continual learning. Indeed, it has been shown empirically by Liang et al. [2019] and Thanh-Tung and Tran [2020] that GANs suffer from catastrophic forgetting, and they adapted CL methods such as Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017] and Synaptic Intelligence (SI) [Zenke, Poole, and Ganguli, 2017] to alleviate forgetting in GANs . However, as we pointed out before, these methods were initially designed with implicit assumptions that a multitask solution always exists and precise task boundaries are available, which make them unsuitable for the setting of multi-agent RL games and GANs.

In this chapter, we propose a CL framework that does not make these assumptions and is applicable in a task agnostic scenario where the tasks can potentially be conflicting with each other. Furthermore, to evaluate our framework, we shift our focus from less forgetting to faster remembering: to rapidly recover the performance on a previously learned task, given the right context as a cue.

4.2 FORMAL STATEMENT

We consider an online learning scenario similar to [Hochreiter, Younger, and Conwell, 2001; Nagabandi, Finn, and Levine, 2019], where at each time step t, a model $\hat{f}$ parametrised by θ_t receives an observation x_t and makes a prediction $\hat{y}_t := \hat{f}(x_t; \theta_t)$. It then gets the ground truth y_t on that task, which can be used to optimize its parameters for better performance in the future. If the data distribution is non-stationary (for example, (x, y) are sampled from task A for a while, then the task switches to task B at some moment t'), then training on the new data might lead to catastrophic forgetting – the new parameters θ' can solve task B but not task A anymore.

Many continual learning methods were proposed to alleviate the problem of catastrophic forgetting. Most of them require either the task identities (A and B in the example) or at least the moment when the task switches (t' in this case). This information, however, is not available when the ground truth y_t depends not only on the observation x_t but also on some hidden task (or context) variable T_t: $y_t = f(x_t, T_t)$, a common situation in partially observable environments [Cassandra, Kaelbling, and Littman, 1994; Monahan, 1982]. Only recently, the CL community started to look at the task agnostic setting [Aljundi et al., 2019; Zeno et al., 2018]. However, all these methods have the underlying assumption that no matter what tasks the learner has been learning, at any time t, it is always possible to find parameters θ_t that fit all previous tasks: $\exists \theta_t \text{ s.t. } \forall t' \leqslant t, \hat{f}(x_{t'}, \theta_t) \approx y_{t'}$. As discussed in the previous section, this assumption does not hold in many realistic scenarios where different tasks conflict with each other: $f(x_t, T_t) \neq f(x_{t'}, T_{t'})$ even when $x_t = x_{t'}$. It follows that, in those settings, catastrophic forgetting cannot be avoided if the model $\hat{f}(\cdot; \theta_t)$ does not depend on the hidden task variable T_t.

4.3 WHAT & HOW FRAMEWORK

Here we propose a framework for task agnostic continual learning that explicitly infers the current task from some context data $\mathcal{D}_t^{\text{ctx}}$ and makes predictions based on both the inputs x_t and the inferred task representations c_t. The framework consists of two modules: a task inference encoder algorithm $\mathcal{F}^{\text{what}} : \mathcal{D}_t^{\text{ctx}} \rightarrow c_t$ that predicts the current task representation c_t based on the context data $\mathcal{D}_t^{\text{ctx}}$, and a decoder algorithm $\mathcal{F}^{\text{How}} : c_t \rightarrow \hat{f}_t$ that maps the task representation c_t to a task specific model $\hat{f}_t : x \rightarrow \hat{y}$.

Under this framework, even when the inputs x_t and $x_{t'}$ are the same, the predictions $\hat{y}_t$ and $\hat{y}_{t'}$ can be different from each other depending on the context. In this work, we choose the recent k observations $\{(x_{t-k}, y_{t-k}), \cdots (x_{t-1}, y_{t-1})\}$ as the context dataset $\mathcal{D}_t^{\text{ctx}}$. This choice is reasonable in an environment where the task variable T_t is piece-wise stationary or changes smoothly. An overview of this framework is illustrated in Figure 4.1.

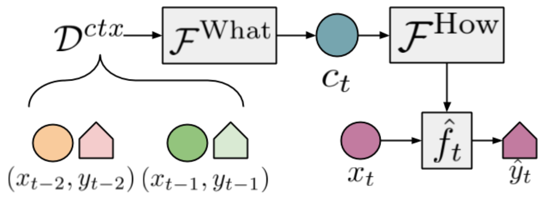


Figure 4.1: What & How framework

4.3.1 *Meta Learning as Task Inference*

In fact, many recently proposed meta-learning methods can be seen as decomposing the problem into What and How modules. For example, Conditional Neural Processes (CNP) [Garnelo et al., 2018] embed the observation and target pairs in context data $(x_i, y_i) \in \mathcal{D}_t^{\text{ctx}}$ by an encoder network $r_i = h(x_i, y_i; \theta_h)$. The embeddings are then aggregated by a commutative operation $\oplus$ (such as the mean operation) to obtain a single embedding of the context: $r_t = \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{ctx}}; \theta_h) = \bigoplus_{x_i, y_i \in \mathcal{D}_t^{\text{ctx}}} h(x_i, y_i; \theta_h)$. At inference time, the context embedding is passed as an additional input to a decoder g to produce the conditional outputs: $\mathcal{F}^{\text{How}}(r_t) = g(\cdot, r_t; \theta_g)$.

Model-Agnostic Meta-Learning (MAML) [Finn, Abbeel, and Levine, 2017] infers the current task by applying one or a few steps of gradient descent on the context data $\mathcal{D}_t^{\text{ctx}}$. In this case, the gradient descent algorithm is the What encoder and the resulting task-specific parameters can be considered a high-dimensional representation of the current task: $\theta_t^k = \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{ctx}}; \theta^{\text{init}}) = U^k(\theta^{\text{init}}, \mathcal{D}_t^{\text{ctx}}, \lambda^{\text{in}}) := \theta_t^{k-1} - \lambda^{\text{in}} \nabla_\theta \mathcal{L}^{\text{in}}(\hat{f}(\cdot; \theta_t^{k-1}), \mathcal{D}_t^{\text{ctx}})$, where the meta parameters θ_t^{init} are the initial values of the model parameters, U^k is the operator that updates θ^{init} by k steps of gradient descent on the context data $\mathcal{D}_t^{\text{ctx}}$ with an inner loop learning rate λ^{in} and an inner loop loss function $\mathcal{L}^{\text{in}}$. The How decoder of MAML returns the task-specific model by simply re-parametrizing the model $\hat{f}$ with θ_t: $\mathcal{F}^{\text{How}}(\theta_t) := \hat{f}(\cdot; \theta_t)$.

Rusu et al. [2019] proposed Latent Embedding Optimization (LEO) which combines the encoder/decoder structure with the idea of inner loop fine-tuning from MAML. The latent task embedding z_t is first sampled from a Gaussian distribution $\mathcal{N}(\mu_t^e, diag(\sigma_t^{e2}))$ whose mean μ_t^e and variance σ_t^{e2} are generated by averaging the outputs of a relation network: $\mu_t^e, \sigma_t^e = \frac{1}{|\mathcal{D}^{\text{cxt}}|^2} \sum_{x_i \in \mathcal{D}^{\text{cxt}}} \sum_{x_j \in \mathcal{D}^{\text{cxt}}} g_r(g_e(x_i), g_e(x_j))$, where $g_r(\cdot)$ is a relation network and $g_e(\cdot)$ is an encoder. Task-dependent weights can then be sampled from a decoder $g_d(\cdot)$: $w_t \sim \mathcal{N}(\mu_t^d, \mathrm{diag}({\sigma_t^d}^2))$, where $\mu_t^d, \sigma_t^d = g_d(z_t)$. The final task representation is obtained by a few steps of gradient descent: $z'_t = \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{cxt}}) := z_t - \lambda^{\text{in}} \nabla_{z'} \mathcal{L}^{\text{in}}(\hat{f}(\cdot; w_t), \mathcal{D}_t^{\text{cxt}})$, and the final task specific weights w'_t are decoded from z': $\mathcal{F}^{\text{How}}(z'_t) = w'_t \sim \mathcal{N}(\mu_t^{d\prime}, \mathrm{diag}({\sigma_t^{d\prime}}^2))$, where $\mu_t^{d\prime}, \sigma_t^{d\prime} = g_d(z'_t)$.

In Fast Context Adaptation via Meta-Learning (CAVIA) [Zintgraf et al., 2019], a neural network model $\hat{f}$ takes a context vector c_t as an additional input: $\hat{y} = \hat{f}(x, c_t; \theta)$. The context vector is inferred from context data by a few steps of gradient descent: $c_t = \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{ctx}}; \theta) := c^{\text{init}} - \lambda^{\text{in}} \nabla_c \mathcal{L}^{\text{in}}(\hat{f}(\cdot, c; \theta), \mathcal{D}_t^{\text{ctx}})$. Then a context-dependent model is returned by the How decoder: $\mathcal{F}^{\text{How}}(c_t) := \hat{f}(\cdot, c_t; \theta)$.

Table 4.1 shows how some of the meta-learning methods can be considered as consisting of What and How modules.

Table 4.1: The What & How modules of some meta learning methods.

Methods	$c_t := \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{cxt}})$	$\mathcal{F}^{\text{How}}(c_t)$
MAML	$\theta_t := \theta_t^{\text{init}} - \lambda^{\text{in}} \nabla_\theta \mathcal{L}^{\text{in}}(\hat{f}(\cdot; \theta), \mathcal{D}_t^{\text{cxt}})$	$\hat{f}(\cdot; \theta_t)$
CNP	$r_t := \bigoplus_{x_i, y_i \in \mathcal{D}_t^{\text{cxt}}} h_\theta(x_i, y_i)$	$g_\theta(\cdot, r_t)$
LEO	$z'_t := z_t - \lambda^{\text{in}} \nabla_{z'} \mathcal{L}^{\text{in}}(\hat{f}(\cdot; w_t), \mathcal{D}_t^{\text{cxt}})$	$w'_t \sim \mathcal{N}(\mu_t^{d\prime}(z'_t), \mathrm{diag}(\sigma_t^{d\prime}(z'_t)^2))$
CAVIA	$c_t := c^{\text{init}} - \lambda^{\text{in}} \nabla_c \mathcal{L}^{\text{in}}(\hat{f}(\cdot, c; \theta), \mathcal{D}_t^{\text{cxt}})$	$\hat{f}(\cdot, c_t; \theta)$

In this work, to showcase our framework, we choose a simple meta learning method called Reptile [Nichol, Achiam, and Schulman, 2018], mainly for its simplicity and for being computationally inexpensive as it does not require second order gradients. Similar to MAML, Reptile tries to learn an initialization of model parameters θ_t^{init} such that optimization on a test task is fast, so its What encoder and How decoder are exactly the same as those of MAML. To update the meta parameters θ_t^{init}, Reptile simply uses the difference between the task-specific parameters and the initialization as the gradient direction: $g_{\theta^{\text{init}}} := \theta_t^{\text{init}} - \theta_t^k = \theta_t^{\text{init}} - U^k(\theta_t^{\text{init}}, \mathcal{D}_t^{\text{ctx}}, \lambda^{\text{in}})$.

In 4.7.3, we also tested full MAML and other meta learning instantiations of the framework, and it can be seen that our framework does not depend on the particular meta learning implementation.

4.3.2 *Continual Meta Learning*

In order to train a meta-learning model, one normally needs access to a task distribution so that i.i.d task samples are available at the same time during training. This is not possible in the online learning setting where tasks are presented sequentially one after the other. Finn et al. [2019] proposed an online meta-learning algorithm called *follow the meta leader* (FTML) based on the framework of regret-minimization. However, FTML requires task information, which is not available in our scenario. In addition, the goal of FTML is faster adaptation with fewer data rather than avoiding catastrophic forgetting, so it simply stores all datapoints from previous tasks, which is considered infeasible in continual learning due to limited resources or privacy reasons. In this work, we choose an alternative framework of online learning called online variational Bayes [Minka, Xiang, and Qi, 2009; Opper, 1998], since it does not require unbounded computational and memory budget. Furthermore, when additional memory budget are available for storing datapoints, online variational Bayes can also be extended by combining it with memory-based online learning methods [Kurle et al., 2020; Minka, Xiang, and Qi, 2009; Nguyen et al., 2018]. In this work, we focus on an algorithm that does not have a growing memory cost over time.

Formally, consider the meta functions $\mathcal{F}^{\text{What}}: \mathcal{D}_t^{\text{ctx}} \to c_t$ and $\mathcal{F}^{\text{How}}: c_t \to \hat{f}_t$, whose composition results in a meta function that maps a context dataset $\mathcal{D}_t^{\text{ctx}}$ to its corresponding task-specific model $\hat{f}_t = \mathcal{F}^{\text{How}} \circ \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{ctx}}; \phi)$, where ϕ represents the collection of all meta parameters. For instance, in MAML and Reptile, $\mathcal{F}^{\text{What}}$ finetunes initial weights θ^{init} on the context dataset $\mathcal{D}_t^{\text{ctx}}$ for k steps to get final weights θ_t^k as the task representation $c_t = \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{ctx}}; \theta^{\text{init}}) = \theta_t^k := \theta_t^{k-1} - \lambda^{\text{in}} \nabla_\theta \mathcal{L}^{\text{in}}(\hat{f}(\cdot; \theta_t^{k-1}), \mathcal{D}_t^{\text{ctx}})$, where $\theta_t^0 := \theta^{\text{init}}$. On the other hand, $\mathcal{F}^{\text{How}}$ simply parameterizes the network with the final weights as the task-specific model: $\mathcal{F}^{\text{How}}(c_t) = \mathcal{F}^{\text{How}}(\theta_t^k) = \hat{f}(\cdot; \theta_t)$. So in the cases of MAML and Reptile, the meta parameters of $\mathcal{F}^{\text{What}}$ are θ^{init} and $\mathcal{F}^{\text{How}}$ does not have meta parameters, thus ϕ is simply θ^{init}.

Using Bayes rule, the posterior $p(\phi|\mathcal{D}_{0:t})$ can be recursively updated by

$$p(\phi|\mathcal{D}_{0:t}) = \frac{p(\mathcal{D}_t|\phi, \mathcal{D}_{0:t-1}) p(\phi|\mathcal{D}_{0:t-1})}{p(\mathcal{D}_t|\mathcal{D}_{0:t-1})}, \tag{4.1}$$

where $\mathcal{D}_t = \{(x_t, y_t)\}$ and $\mathcal{D}_{0:t}$ is the union of all datasets up to t.

In addition, we make two assumptions: first, the input x_t is independent of the meta parameters ϕ given previous data $\mathcal{D}_{0:t-1}$: $p(x_t|\phi, \mathcal{D}_{0:t-1}) = p(x_t|\mathcal{D}_{0:t-1})$; second, a moving window of context data is informative about the task variable: $p(\mathcal{D}_t|\phi, \mathcal{D}_{0:t-1}) = p(\mathcal{D}_t|\phi, \mathcal{D}_t^{\text{ctx}})$. Under these assumptions, the posterior can be further simplified:

$$p(\phi|\mathcal{D}_{0:t}) = \frac{p(y_t|x_t, \phi, \mathcal{D}_t^{\text{ctx}}) p(\phi|\mathcal{D}_{0:t-1})}{p(y_t|x_t, \mathcal{D}_{0:t-1})}, \tag{4.2}$$

where the likelihood term $p(y_t|x_t, \phi, \mathcal{D}_t^{\text{ctx}})$ is modeled by the What and How framework:

$$p(y_t|x_t, \phi, \mathcal{D}_t^{\text{ctx}}) = p(y_t|\hat{f}_t(x_t)) = p(y_t|\mathcal{F}^{\text{How}} \circ \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{ctx}}; \phi)(x_t)). \tag{4.3}$$

In online variational Bayes, the true posterior $p(\phi|\mathcal{D}_{0:t})$ is approximated by a parametric distribution $q_t(\phi)$ by minimizing the Kullback-Leibler divergence

$$q_t(\phi) = \arg\min_{q(\phi)} \mathrm{KL}(q(\phi)||p(\phi|\mathcal{D}_{0:t})). \tag{4.4}$$

Using Eq. 4.2, this KL divergence can be written as:

$$\begin{aligned}\mathrm{KL}(q(\phi)||p(\phi|\mathcal{D}_{0:t})) =& \mathbb{E}_{q(\phi)}\left[\log \frac{q(\phi)}{p(\phi|\mathcal{D}_{0:t})}\right]\\ =& \mathbb{E}_{q(\phi)}\left[\log \frac{q(\phi)p(y_t|x_t,\mathcal{D}_{0:t-1})}{p(y_t|x_t,\phi,\mathcal{D}_t^{\mathrm{ctx}})p(\phi|\mathcal{D}_{0:t-1})}\right]\\ =& \mathrm{KL}(q(\phi)||p(\phi|\mathcal{D}_{0:t-1})) - \mathbb{E}_{q(\phi)}\left[\log p(y_t|x_t,\phi,\mathcal{D}_t^{\mathrm{ctx}})\right]\\ &+ \log p(y_t|x_t,\mathcal{D}_{0:t-1}).\end{aligned} \tag{4.5}$$

Since $\log p(y_t|x_t,\mathcal{D}_{0:t-1})$ does not depend on $q(\phi)$, it can be omitted in the optimization. Furthermore, if we use a parametric distribution at every time step, then $p(\phi|\mathcal{D}_{0:t-1})$ can be approximated by $q_{t-1}(\phi)$ and the minimization problem in Eqn. 4.4 is equivalent to maximizing the evidence lower bound (ELBO) $\mathcal{E}(q(\phi),\mathcal{D}_{0:t},q_{t-1}(\phi))) := \mathbb{E}_{q(\phi)}[\log p(y_t|x_t,\phi,\mathcal{D}_t^{\mathrm{ctx}})] - \mathrm{KL}(q(\phi)||q_{t-1}(\phi))$:

$$\begin{aligned} q_t(\phi) =& \arg\max_{q(\phi)} \mathcal{E}(q(\phi),\mathcal{D}_{0:t},q_{t-1}(\phi)) &(4.6)\\ =& \arg\max_{q(\phi)} \mathbb{E}_{q(\phi)}[\log p(y_t|x_t,\phi,\mathcal{D}_t^{\mathrm{ctx}})] - \mathrm{KL}(q(\phi)||q_{t-1}(\phi)). &(4.7)\end{aligned}$$

In this work, we choose the parametric distribution to be a factorized Gaussian $q_t(\phi) = \prod_i \mathcal{N}(\phi_i|\mu_i(t),\sigma_i(t))$, where ϕ_i is the i-th component of ϕ. Using the reparametrization trick [Kingma and Welling, 2013]: $\phi_i = \mu_i + \sigma_i\epsilon_i,\ \epsilon_i \sim \mathcal{N}(0,1)$ and let $q_{t-1}(\phi) = \prod_i \mathcal{N}(\phi_i|\mu_i(t-1),\sigma_i(t-1))$, we can find the maximum of the ELBO by solving the following equations

$$\begin{aligned}\frac{\partial}{\partial\mu_i(t)}\mathcal{E}(q(\phi),\mathcal{D}_{0:t},q_{t-1}(\phi)) = 0, &\quad(4.8)\\ \frac{\partial}{\partial\sigma_i(t)}\mathcal{E}(q(\phi),\mathcal{D}_{0:t},q_{t-1}(\phi)) = 0, &\quad(4.9)\end{aligned}$$

and the results are update rules for μ_i and σ_i that are similar to Bayesian Online Learning [Opper, 1998] and Bayesian Gradient Descent (BGD) [Zeno et al., 2018], but on the meta-level (see Appendix 4.7.1 for details of the derivation):

$$\mu_i(t) = \mu_i(t-1) - \sigma_i^2(t-1)\mathbb{E}_{p(\epsilon)}[\frac{\partial \mathcal{L}_t(\phi)}{\partial \phi_i}], \tag{4.10}$$

$$\sigma_i(t) = \sigma_i(t-1)\sqrt{1 + \Big(\frac{1}{2}\sigma_i(t-1)\mathbb{E}_{p(\epsilon)}[\frac{\partial \mathcal{L}_t(\phi)}{\partial \phi_i}\epsilon_i]\Big)^2} - \frac{1}{2}\sigma_i^2(t-1)\mathbb{E}_{p(\epsilon)}[\frac{\partial \mathcal{L}_t(\phi)}{\partial \phi_i}\epsilon_i], \tag{4.11}$$

where $\mathcal{L}_t(\phi) = -\log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})$. An intuitive interpretation of these learning rules is that weights μ_i with smaller uncertainty σ_i are more important for the knowledge accumulated so far, thus they should change slower in the future in order to preserve the learned knowledge.

In practice, we introduce learning rates for both 4.10 and 4.11. Maximum a posterior (MAP) estimate of ϕ is used for prediction. We approximate the expectation in the μ update rule 4.10 by the gradient at the mean, and for the expectation in the σ update rule 4.11, we estimate it by Monte Carlo sampling method. The final algorithm called W&H is described in Algorithm 3.

COMPLEXITY The number of parameters required by the W&H algorithm is 3 times that of the base model, since it needs to store the mean and the standard deviation of the initialization and a copy of the current task-specific parameters. In terms of time complexity, the computation of the mean update Δ_μ and the Monte Carlo (MC) sample of the standard deviation update Δ_σ^j can be parallelized. In that case, W&H has only constant computational overhead compared to Reptile due to sampling and gradient averaging. If the MC sampling process is implemented sequentially, the total time complexity is $\mathcal{O}(J+1)$ times that of the Reptile algorithm, where J is the number of sampling steps. Similar to the findings in Zeno et al. [2018], we find that in practice the number of MC samples has negligible effect on the performance of the algorithm. So we used only a single sample for our experiments.

4.4 RELATED WORK

CONTINUAL LEARNING While effective at preventing forgetting, most CL methods we reviewed in Chapter 1 either rely on knowledge of task boundaries or require task labels to select a sub-module for adaptation and prediction, hence cannot be directly applied in the task agnostic scenario considered here. To circumvent this issue, Kirkpatrick et al. [2017] used Forget-Me-Not (FMN) [Milan et al., 2016] to detect task boundaries and combined it with EWC to consolidate memory when task switches.

Algorithm 3 What & How (using Reptile)

```
**Input:** $\lambda^{in}, k, J, \mathcal{D}_0^{ctx}, \eta_\mu, \eta_\sigma$ and initial values of $\mu, \sigma$
**for** $t = 0, 1, \ldots$ **do**
    $\theta_t \leftarrow U^k(\mu; \mathcal{D}_t^{ctx}, \lambda^{in}) = \mathcal{F}^{What}(\mathcal{D}_t^{ctx}; \mu)$,
    $\hat{y}_t \leftarrow \hat{f}(x_t; \theta_t) = \mathcal{F}^{How}(\theta_t)(x_t)$
    $\Delta_\mu \leftarrow \eta_\mu \frac{\partial \mathcal{L}_t(\phi)}{\partial \phi}\Big|_{\phi=\mu} \approx \eta_\mu(\mu - U^k(\mu, \mathcal{D}_t^{ctx}, \lambda^{in}))$
    **for** $j = 1$ **to** $J$ **do**
        $\epsilon^j \sim \mathcal{N}(0, 1)$
        $\phi^j = \epsilon^j \sigma + \mu$
        $\Delta_\sigma^j \leftarrow \epsilon^j \frac{\partial \mathcal{L}_t(\phi)}{\partial \phi}\Big|_{\phi=\phi^j} \approx \epsilon^j(\phi^j - U^k(\phi^j, \mathcal{D}_t^{ctx}, \lambda^{in}))$
    **end for**
    $\Delta_\sigma \leftarrow \eta_\sigma \frac{1}{J} \sum_{j=1} \Delta_\sigma^j$
    $\mu_i \leftarrow \mu_i - \sigma_i^2 \Delta_{\mu_i}$
    $\sigma_i \leftarrow \sigma_i \sqrt{1 + \left(\frac{1}{2}\sigma_i \Delta_{\sigma_i}\right)^2} - \frac{1}{2}\sigma_i^2 \Delta_{\sigma_i}$,
    Update $\mathcal{D}_t^{ctx}$ with $\{(x_t, y_t)\}$ to get $\mathcal{D}_{t+1}^{ctx}$
**end for**
```

However, FMN requires a generative model that computes exact data likelihood, which limits it from scaling to complex tasks. Bayesian Gradient Descent (BGD) [Zeno et al., 2018], as we discussed before, adopts the framework of online variational Bayes, and approximates the posterior with a diagonal Gaussian distribution. More recently, Aljundi et al. [2019] proposed a rehearsal-based method to select a finite number of data that are representative of all data seen so far. All of these methods assume that it is possible to learn one model that fits all previous data, neglecting the scenario where different tasks may conflict with each other, hence do not allow task-specific adaptations.

META LEARNING As with continual learning, different families of approaches exist for meta-learning. **Memory-based methods** [Santoro et al., 2016] rely on a recurrent model (optimizer) such as LSTM to learn a history-dependent update function for the lower-level learner (optimizee). Andrychowicz et al. [2016] trained an LSTM to replace the stochastic gradient descent algorithm by minimizing the sum of losses of the optimizees on multiple prior tasks. Ravi and Larochelle [2017] use an LSTM-based meta-learner to transform the gradient and loss of the base-learners on every new example to the final updates of the model parameters. **Metric-based methods** learn an embedding space in which new tasks can be solved efficiently. Koch, Zemel, Salakhutdinov, et al. [2015] trained siamese networks to tell if two images are similar by converting the distance between their feature embeddings to the probability of whether they are from the same class. Vinyals et al. [2016] proposed the matching network to improve the embeddings of a test image and the support images by taking the entire support set as context input. The approaches discussed in Section 4.3.1 instead belong to the family of **optimization-based meta-learning** methods, which learn to adjust the gradient descent optimization process itself. In this category, beside the online meta learning method [Finn et al., 2019] discussed before, the most relevant work is from Nagabandi, Finn, and Levine, 2019, who studied fast adaptation in a non-stationary environment by learning an ensemble of networks, one for each task. Unlike our framework, they applied meta learning for initialization of new networks in the ensemble instead of for task inference. A drawback of this approach is that the size of the ensemble grows over time and is unbounded, hence can become memory-consuming when there are many tasks.

4.5 EXPERIMENTS

We design a series of experiments to thoroughly evaluate the effectiveness of the What & How framework, and compare it to BGD and other CL methods (EWC, online EWC, SI, LwF, DGR, DGR+Distill) implemented by Ven and Tolias [2019]. We also include the following baselines: **None**: the model is trained sequentially using Adam [Kingma and Ba, 2014] in the standard way without applying any CL method; **Joint**: all tasks seen so far are trained at the same time, this scheme usually sets the upper bound for CL methods. **FTML**: a non-CL method that trains a meta model by uniformly sampling from previously encountered tasks, therefore it requires task labels and stores all data. Implementation details can be found in the Appendix.

4.5.1 *Label-Permuted MNIST*

In this experiment, we first create a different permutation of 10 classes for every task, with which we shuffle the classes in the labels. For instance, digit 0 might be the first class in one task but the second class in another task. The reason for this design is to ensure that a multi-task solution does not exist since the network has to map the same image to different labels for different tasks. In this way, we can test whether our framework is able to quickly adapt its behavior according to the current context. Five tasks are created with this method and are presented sequentially for 1000 iterations each to an MLP with 2 hidden layers of 1000 neurons. In each iteration, a mini-batch of 128 images is presented to the network. Note that except BGD, None, Joint and our method, all the other methods simply cannot be directly applied in the task agnostic

Table 4.2: Zero-shot and few-shot recall (with 128 context examples) accuracies for different methods on the label-permuted MNIST tasks. The results are average accuracies over all tasks. The mean and the standard error of mean (SEM) are computed over 5 runs with different random seeds for each method. The first three methods are non-CL baselines.

Methods	Zero-shot	Few-shot (k=5)	Few-shot (k=200)
None	28.07 ± 1.19	27.65 ± 0.79	82.37 ± 0.58
Joint	35.74 ± 0.99	33.51 ± 1.81	85.29 ± 0.40
FTML	28.85 ± 1.32	97.99 ± 0.03	98.38 ± 0.04
BGD	28.18 ± 1.17	70.23 ± 1.24	85.27 ± 0.53
EWC	27.86 ± 1.19	29.49 ± 0.90	82.00 ± 0.55
Online EWC	27.85 ± 1.20	30.32 ± 0.70	82.45 ± 0.60
SI	28.06 ± 1.19	29.62 ± 0.97	82.97 ± 0.33
LwF	**33.70 ± 1.02**	32.66 ± 1.18	83.94 ± 0.40
DGR	27.88 ± 1.12	23.40 ± 1.04	84.08 ± 0.25
DGR+Distill	28.03 ± 1.14	28.97 ± 1.14	83.09 ± 0.34
W&H (Ours)	28.16 ± 1.19	**74.27 ± 0.56**	**92.05 ± 0.14**

scenario, so we provide the necessary task information for these methods in order to perform the comparison.

ZERO-SHOT VS. FEW-SHOT RECALL At the end of the entire learning process, we test the learner's classification accuracy of each task in two ways. The first way is to directly apply the final model on the testing data without any adaptations, this corresponds to the *zero-shot recall* accuracy traditionally used in CL. For the W&H algorithm, this means we apply the model simply with the learned initialization. In the second way, we provide the final model with a mini-batch of 128 images from the training set of each task as context data, the model is fine-tuned on the context data for k steps before it is tested on the testing data of that task. We call this metric *few-shot recall* accuracy. We report our results for two different number of fine tuning steps $k = 5$ and $k = 200$ as they demonstrate different properties of the compared methods. When $k = 5$, not all methods have converged, so the few-shot accuracy shows how fast a method adapts to the context data, whereas when $k = 200$, all methods have converged, so the few-shot accuracy shows how good the final solution is.

Table 4.3: Average test accuracy (over all 10 tasks) on the permuted MNIST experiment. The accuracies for BGD and *W*&H reported here are the mean (± SEM) over 5 runs. Other results are taken directly from Ven and Tolias [2019]

Approaches	**Methods**	**Domain-IL**
Baselines	None	78.51 ± 0.24
	Joint	97.59 ± 0.01
	FTML	97.47 ± 0.02
Regularization	EWC	94.31 ± 0.11
	Online EWC	94.42 ± 0.13
	SI	95.33 ± 0.11
Replay	LwF	72.64 ± 0.52
	DGR	95.09 ± 0.04
	DGR+Distill	**97.35 ± 0.02**
Online Bayesian	BGD	93.02 ± 0.33
	W&H (Ours)	93.37 ± 0.33

Table 4.2 summarizes the performance of our method and other baselines. It can be seen that in this experiment, it is impossible to achieve good performance with zero-shot recall since a multi-task solution does not exist. Even the joint training scheme which is considered the upper bound for CL achieves very poor accuracies. In the few-shot setting, the upper bound is the FTML method, since it has access to all previous data and task information. Among the CL methods, our framework significantly outperforms the other baselines, even without access to any task information during training and without storing data from the past.

4.5.2 *Permuted and Split MNIST*

We also test the What & How framework on the standard CL benchmarks called Permuted MNIST [Goodfellow et al., 2014a] and Split MNIST [Zenke, Poole, and Ganguli, 2017]. In these experiments, tasks are not conflicting with each other, so multi-task solutions do exist. Ven and Tolias [2019] described three scenarios for these experiments based on what task information are available at test time: **task-incremental learning**: models are always informed about which task is presented; **domain-incremental learn-**

Table 4.4: Average test accuracy (over all 5 tasks) on the split MNIST experiment. The accuracies for BGD and W&H reported here are the mean (± SEM) over 5 runs. Other results are taken directly from Ven and Tolias [2019]

Approaches	**Methods**	**Domain-IL**
Baselines	None	59.21 ± 2.04
	Joint	98.42 ± 0.06
	FTML	98.40 ± 0.07
Regularization	EWC	63.95 ± 1.90
	Online EWC	64.32 ± 1.90
	SI	65.36 ± 1.57
Replay	LwF	71.50 ± 1.63
	DGR	95.72 ± 0.25
	DGR+Distill	$\mathbf{96.83 \pm 0.20}$
Online Bayesian	BGD	66.07 ± 2.13
	W&H (Ours)	67.33 ± 2.03

ing: task ID is not provided at test time; **class-incremental learning**: task ID is not provided and should be inferred at test time. However, they assumed that *during training* there are clear and well-defined task boundaries available for the learner. Since our focus is task agnostic CL during training, we only consider the *domain-incremental scenario* at test time, because in the other two scenarios, the task boundaries are anyways available during training (in the class-incremental case, a class corresponds to a task, which can be simply detected from the labels), it is not necessary to apply a task-agnostic method.

In the permuted MNIST protocol, a new task is created by shuffling the pixels of all images in MNIST by a fixed permutation. We present 10 such tasks sequentially for 5000 iterations each. After all tasks are learned, the network has to predict the digit from an image without knowing the permutation. For the split MNIST protocol, the original MNIST dataset is divided into 5 subsets with 2 digits each. We present one of these subsets at a time for 2000 iterations. At the end of the training, the network has to predict if an image was the first class or the second class in its subset, without knowing which subset the image is from. To be comparable with the results from Ven and Tolias [2019], we used exactly the same network architecture, experiment setup and hyper-parameters for these two experiments.

The results are displayed in Table 4.3 and Table 4.4. Again, BGD and *our method do not have access to any task information*, while the other methods cannot be directly applied without task information. For our method, we use the previous mini-batch as context data during training, and at testing time, the learned initialization was directly used without any context-dependent adaption, it can be seen from the tables that our method performs the same as BGD, which also has similar performance to the regularization-based task-aware methods. This means our framework is also applicable when multi-task solutions exist.

4.5.3 *Sine Regression*

One desideratum of CL is the ability of *forward transfer*. Lopez-Paz et al. [2017] defined a metric for forward transfer based on a model's "zero-shot" performance on a future task. We generalize the concept of forward transfer to the "few-shot" setting: positive forward transfer should allow a model to learn faster on a future task similar to the ones it has learned. This definition coincides with generalization in meta-learning, which refers to how fast a meta-learner can learn a new task at meta-test time. We show that our method is capable of forward transfer by comparing it to SGD, BGD, Reptile on the sine regression tasks commonly used in meta learning literature. We randomly generate 100 different sine curves and present them sequentially to the learners for 500 iterations each (details in the Appendix). At the end of the learning process, we evaluate the few-shot performance (in terms of mean squared errors) of all learners on these 100 sine curves as well as 100 new sine curves *it has not seen before*. In particular, the learners are evaluated after they are adapted on 10 context datapoints until convergence. Table 4.5 summarizes the results of this experiment. Except for the upper bound baseline FTML, the other methods perform poorly due to either catastrophic forgetting or no multi-task solution.

4.5.4 *Continual GAN*

Finally, we apply our framework in the setting of GAN training, where the ultimate goal is to find the optimal parameters θ_G^* for a generator $G(z;\theta_G)$ by optimizing a minimax objective [e.g. Goodfellow et al., 2014b; Metz et al., 2016]:

$$\theta_G^* = \operatorname*{argmin}_{\theta_G} \max_{\theta_D} l(\theta_G, \theta_D) = \operatorname*{argmin}_{\theta_G} l(\theta_G, \theta_D^*(\theta_G)), \tag{4.12}$$

where $\theta_D^*(\theta_G) := \operatorname{argmax}_{\theta_D} l(\theta_G, \theta_D)$ and $l(\theta_G, \theta_D) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x;\theta_D))] + \mathbb{E}_{z\sim\mathcal{N}(0,1)}[\log(1 - D(G(z;\theta_G);\theta_D))]$.

Following our discussion at the beginning of this section, continually learning $D(\cdot;\theta_D)$ without forgetting is impossible, because at different moments t and t' during training,

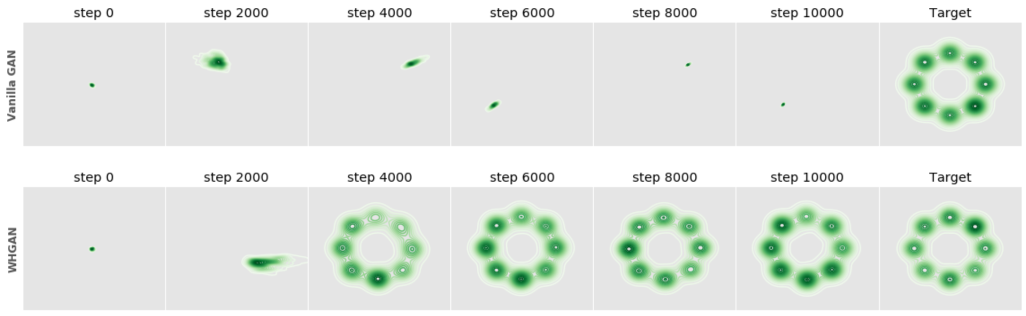


Figure 4.2: The What and How framework prevents mode collapse on the 2D mixture of Gaussians dataset. The first six columns show KDE plots of 512 samples from the generator at different training steps. The last column are created from 512 samples of the real distribution. The first row shows standard training of a vanilla GAN. The second row shows the same GAN with the discriminator trained by the What and How framework.

the generator distributions may be different ($p_G \neq p_{G'}$), hence the corresponding optimal discriminator parameters cannot be the same: $\theta^*_D(\theta_G) \neq \theta^*_D(\theta_{G'})$. In other words, the tasks for the discriminator at time t and t' are conflicting with each other. Adopting the What & How framework, we focus on continually learning a model for $\theta^*_D(\cdot)$ in Eq.4.12 instead of the discriminator $D(\cdot)$. This model corresponds to our What encoder: given a set of data points $\mathcal{D}_G := \{G(z_n; \theta_G)\}_n$ sampled from the current generator as context data, the What encoder is trained to approximate the optimal discriminator parameters $\theta^*_D(\theta_G) \approx \mathcal{F}^{\text{What}}(\mathcal{D}_G; \phi)$ by k steps of inner loop updates on the context data, where the meta-parameters ϕ corresponding to an initialization of the discriminator $\phi := \theta^{\text{init}}_D$ are learned by Alg.3. The generator-specific discriminator returned by the How decoder $\mathcal{F}^{\text{How}} \circ \mathcal{F}^{\text{What}}(\mathcal{D}_G; \phi) := D(\cdot; \mathcal{F}^{\text{What}}(\mathcal{D}_G; \phi))$ is used to compute the loss of θ_G and to update the current generator[1]. This way, the conflict at the level of θ_D is resolved at the level of ϕ: it is possible, in theory, to find a single ϕ that maximizes both $l(\theta_G, \mathcal{F}^{\text{What}}(\mathcal{D}_G; \phi))$ and $l(\theta_{G'}, \mathcal{F}^{\text{What}}(\mathcal{D}_{G'}; \phi))$, even when $\theta_G \neq \theta_{G'}$.

Thanh-Tung and Tran [2020] and Liang et al. [2019] showed that a notorious problem for GAN training called mode collapse [Che et al., 2016] is interrelated with catastrophic forgetting and can cause the training process to never converge, since the generator is always optimized to revisit a mode that the discriminator has forgotten. Therefore, overcoming catastrophic forgetting problem in the discriminator should be able to break

1 Unlike the Unrolled GAN Metz et al., 2016, we do not backprop through the inner loop optimization of the discriminator when we compute the gradients of the generator, even though this can provide more accurate gradients for the generator and further improve the performance of our GANs. The reason is that we want to isolate the effect of our method from that of the Unrolled GAN.

Table 4.5: Average MSE of different methods on the sine curve regression tasks at the end of training. The "Seen" column contains the MSE over 200 sine curves presented during training. The "Unseen" column contains the MSE over 200 new sine curves the learners have not seen before. The results reported here are the mean (± SEM) over 5 trials.

Methods	Seen	Unseen
FTML	0.24 ± 0.04	0.31 ± 0.06
SGD	0.97 ± 0.06	1.50 ± 0.10
BGD	1.01 ± 0.05	1.82 ± 0.06
Reptile	1.02 ± 0.14	1.23 ± 0.17
W&H (Ours)	**0.56 ± 0.02**	**0.78 ± 0.02**

this mode revisiting cycles and reduce mode collapse, as shown in the experiments below.

2D MIXTURE OF GAUSSIANS As we discussed before, in GAN training, the objective function for the discriminator is defined based on the parameters of the generator. So the task for the discriminator actually depends on the parameters of the generator. As we are updating the generator, the task for the discriminator is also gradually changing, hence there are potentially infinite number of tasks and there are no predefined boundaries between different tasks. To directly visualize the effect of our method, we first applied it to train a simple GAN with synthetic data generated from a mixture of Gaussian distributions on 2D space. The network architecture and experiment setup are exactly the same as in [Metz et al., 2016]. We first trained a vanilla GAN with standard techniques on this dataset, the first row (Vanilla GAN) of Figure 4.2 shows that it entered a non-convergent cycle of revisiting the modes. We then applied the What & How method (with $k = 3$ steps in the inner loop) to the discriminator while keeping the rest of the experiment setup and hyper-parameters the same. As can be seen in the second row (WHGAN) of Figure 4.2, although mode collapse also occurred at the beginning of the training process, our networks were able to avoid the repeating cycle, and eventually converged to a distribution covering all modes.

DCGAN ON CIFAR10 In this experiment, we compare the differences between a Deep Convolutional GAN (DCGAN) [Radford, Metz, and Chintala, 2016] trained with and without our method on the CIFAR10 dataset [Krizhevsky, 2009]. Since it is hard to visualize mode collapse for high dimensional image data, we used two bin-based metrics

Table 4.6: Quantitative Evaluation of DCGAN and WHGAN on CIFAR10. The results reported here are the mean (± SEM) over 5 random trials. ↓ (resp. ↑) indicates lower (resp. higher) is better.

Metrics	**DCGAN**	**WHGAN**(Ours)
NDB↓	48.20± 4.68	**26.00 ± 0.89**
JSD↓	0.013± 0.0016	**0.0069 ± 0.00019**
FID↓	47.52± 0.49	46.78 ± 0.63
IS ↑	4.40 ± 0.028	4.47 ± 0.027

called NDB and JSD [Richardson and Weiss, 2018] to evaluate the resulting GANs. To compute NDB, the real samples were first clustered by K-means into $K = 200$ bins, which can be considered as modes of the data distribution. Then $N = 50000$ images sampled from the generator were assigned to their nearest bins. For each bin, a two-sample test was performed to decide if the synthesized samples are statistically different from the real samples. NDB is then simply the *number of statistically different bins* and JSD is the *Jensen-Shannon divergence* between the real distribution and the generator distribution over these bins. JSD is defined by:

$$JSD(P||Q) = \frac{1}{2}KL(P||M) + \frac{1}{2}KL(Q||M), \tag{4.13}$$

where $M = \frac{1}{2}(P + Q)$. Lower NDB and JSD scores imply more similarity between two distributions, and hence less mode collapse. In addition, we also evaluate the resulting GANs with the Inception Score (IS) [Salimans et al., 2016] and the Fréchet Inception Distance (FID) [Heusel et al., 2017], which are metrics based on the image features extracted by the Inception Network [Szegedy et al., 2015]. Higher IS and lower FID indicate better quality of the generated images.

Table 4.6 compares the performance of the original DCGAN and one trained with our framework (WHGAN). Again, we kept the network architecture and all hyper-parameters the same, except that the discriminator in WHGAN was trained with the What & How method. In both cases, we trained the networks for 50000 iterations with a mini-batch size 64. For our method, $k = 3$ steps are used in the inner loop of the What encoder. The results show that WHGAN achieved significantly lower NDB and JSD while maintaining the same image quality. Figure 4.3 are image samples randomly drawn from a DCGAN and a WHGAN trained in this experiment.

4.6 CONCLUSIONS

In this chapter, we showed that when a multi-task solution does not exist and task information is unknown, catastrophic forgetting is inevitable. A framework that can infer task information explicitly from context data was proposed to resolve this problem. The framework separates the inference process into two components: one for representing *What* task is presented, and the other for describing *How* to solve the given task. In addition, our framework unifies many meta learning methods and establishes a connection between continual learning and meta learning, leveraging the advantages of both.

From the meta learning perspective, our framework addresses the continual meta learning problem by applying continual learning techniques on the meta variables, therefore allowing meta knowledge to accumulate over an extended period; from the continual learning perspective, our framework addresses the task agnostic continual learning problem by explicitly inferring the task when the task information is not available and a multi-task solution does not exist. This allows us to shift the focus of continual learning from less forgetting to faster remembering, given the right context.

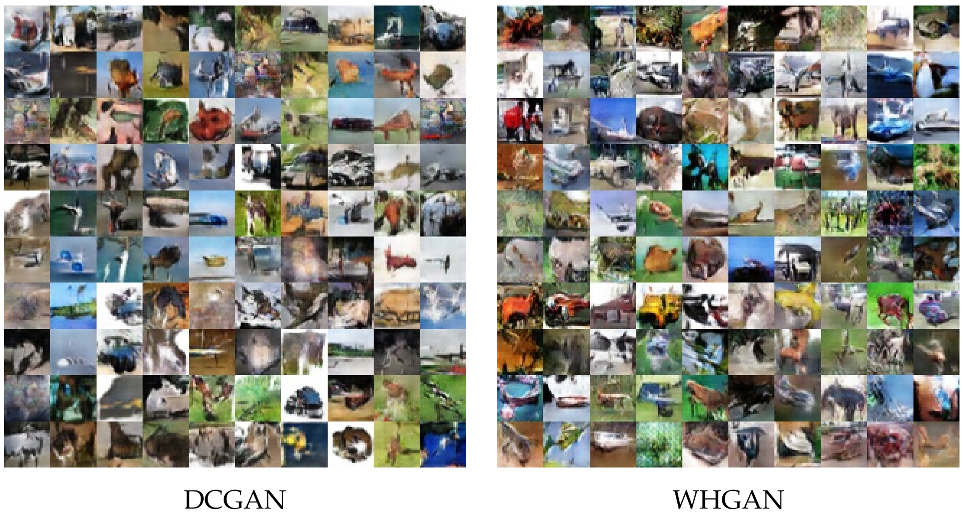


Figure 4.3: Generated images from a DCGAN and a WHGAN trained on CIFAR10.

4.7 APPENDIX

4.7.1 *Update Rule Derivation*

The derivation of the update rules 4.10 and 4.11 is the same as the one given in Zeno et al. [2018]. We repeat them here for ease of reference. We first note that

$$\begin{aligned}\mathcal{E}(q_t(\phi), \mathcal{D}_{0:t}, q_{t-1}(\phi))) :=& \mathbb{E}_{q_t(\phi)}[\log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})] - \mathrm{KL}(q_t(\phi)\|q_{t-1}(\phi)) \\ =& \mathbb{E}_{q_t(\phi)}[f(\phi, \mu(t), \sigma(t))],\end{aligned} \tag{4.14}$$

where we define $f(\phi, \mu(t), \sigma(t)) := \log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx}) + \log q_{t-1}(\phi) - \log q_t(\phi)$ to simplify the formula. Since both $q_{t-1}(\phi)$ and $q_t(\phi)$ are factorized Gaussians, we have

$$\log q_t(\phi) = -\frac{N}{2}\log 2\pi - \sum_{k=1}^{N}\log\sigma_k(t) - \sum_{k=1}^{N}\frac{(\phi_k - \mu_k(t))^2}{2\sigma_k^2(t)}, \tag{4.15}$$

$$\log q_{t-1}(\phi) = -\frac{N}{2}\log 2\pi - \sum_{k=1}^{N}\log\sigma_k(t-1) - \sum_{k=1}^{N}\frac{(\phi_k - \mu_k(t-1))^2}{2\sigma_k^2(t-1)}. \tag{4.16}$$

Rewrite the first-order necessary condition in 4.8 for the optimal $\mu_i(t)$ using the reparametrization trick $\phi_i = \mu_i(t) + \sigma_i(t)\epsilon_i$

$$\begin{aligned}&\frac{\partial}{\partial\mu_i(t)}\mathbb{E}_{q_t(\phi)}[f(\phi, \mu(t), \sigma(t))] \\ =& \mathbb{E}_{p(\epsilon)}[\frac{\partial}{\partial\mu_i(t)}f(\phi, \mu(t), \sigma(t))] \\ =& \mathbb{E}_{p(\epsilon)}[\frac{\partial f(\phi, \mu(t), \sigma(t))}{\partial\phi_i}\frac{\partial\phi_i}{\partial\mu_i(t)} + \frac{\partial f(\phi, \mu(t), \sigma(t))}{\partial\mu_i(t)}] = 0.\end{aligned} \tag{4.17}$$

Computing the derivatives using 4.15, 4.16, we get

$$\begin{aligned}&\mathbb{E}_{p(\epsilon)}[\frac{\partial\log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial\phi_i} - \frac{\phi_i - \mu_i(t-1)}{\sigma_i^2(t-1)} + \frac{\phi_i - \mu_i(t)}{\sigma_i^2(t)} - \frac{\phi_i - \mu_i(t)}{\sigma_i^2(t)}] \\ =& \mathbb{E}_{p(\epsilon)}[\frac{\partial\log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial\phi_i}] - \frac{\mu_i(t) - \mu_i(t-1)}{\sigma_i^2(t-1)} = 0,\end{aligned} \tag{4.18}$$

which implies the mean update rule 4.10:

$$\mu_i(t) = \mu_i(t-1) - \sigma_i^2(t-1)\mathbb{E}_{p(\epsilon)}[-\frac{\partial\log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial\phi_i}]. \tag{4.19}$$

Similar to 4.17, we can write down the first-order necessary condition for the optimal $\sigma_i(t)$

$$\mathbb{E}_{p(\epsilon)}[\frac{\partial f(\phi, \mu(t), \sigma(t))}{\partial \phi_i} \frac{\partial \phi_i}{\partial \sigma_i(t)} + \frac{\partial f(\phi, \mu(t), \sigma(t))}{\partial \sigma_i(t)}] = 0. \tag{4.20}$$

Computing the derivatives leads to the following equation:

$$\begin{aligned}
&\mathbb{E}_{p(\epsilon)}[(\frac{\partial \log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial \phi_i} - \frac{\phi_i - \mu_i(t-1)}{\sigma_i^2(t-1)} + \frac{\phi_i - \mu_i(t)}{\sigma_i^2(t)})\epsilon_i \\
&+\frac{1}{\sigma_i(t)} - \frac{(\phi_i - \mu_i(t))^2}{\sigma_i^3(t)}] \\
=&\mathbb{E}_{p(\epsilon)}[(\frac{\partial \log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial \phi_i}\epsilon_i] - \frac{\sigma_i(t)}{\sigma_i^2(t-1)} + \frac{1}{\sigma_i(t)} = 0,
\end{aligned} \tag{4.21}$$

which is essentially a quadratic equation of $\sigma_i(t)$, solving it for $\sigma_i(t) > 0$ gives us the update rule 4.11:

$$\begin{aligned}
\sigma_i(t) =& \sigma_i(t-1)\sqrt{1 + \left(\frac{1}{2}\sigma_i(t-1)\mathbb{E}_{p(\epsilon)}[\frac{\partial \log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial \phi_i}\epsilon_i]\right)^2} \\
&- \frac{1}{2}\sigma_i^2(t-1)\mathbb{E}_{p(\epsilon)}[\frac{\partial \log p(y_t|x_t, \phi, \mathcal{D}_t^{ctx})}{\partial \phi_i}\epsilon_i].
\end{aligned} \tag{4.22}$$

4.7.2 *Experimental Details*

LABEL-PERMUTED MNIST Each task is a ten-way classification. The original images are zero-padded to 32x32 pixels. A random permutation of 10 classes was generated and applied to all labels. The standard training/test-split was used. For W&H, we used inner loop learning rate $\lambda^{in} = 0.1$, initial standard deviation $\sigma = 0.12$, mean and standard deviation learning rates are $\eta_\mu = 1.$ and $\eta_\sigma = 1.0$, $J = 1$ Monte Carlo (MC) sample was used. For BGD, the initial standard deviation was $\sigma = 0.06$ and the mean learning rate was $\eta_\mu = 1.0$, the number of MC samples was 2. For FTML, SGD with learning rate $\lambda^{in} = 0.1$ was used in the inner loop, and Adam with learning rate 0.001 was used in the outer loop. For the other baselines, their implementation and hyper-parameters were the same as in Ven and Tolias [2019] for the domain-incremental permuted MNIST experiment. To compute the few-shot accuracy, all models were adapted on the 10 context data points for 200 gradient steps to ensure convergence.

PERMUTED AND SPLIT MNIST In order to have a fair comparison, for these two experiments, the setup and the hyper-parameters of all baselines were kept the same as in the domain-incremental scenario in Ven and Tolias [2019].

In permuted MNIST, the original images were zero-padded to 32x32 pixels. Each task is a ten-way classification. For W&H, $k = 5$ steps and a learning rate $\lambda^{in} = 0.1$ were used in the loop during training, initial standard deviation was $\sigma = 0.06$, $\eta_\mu = 1$, $\eta_\sigma = 1$. $J = 1$ MC samples was used. At testing time, no inner loop updates were used. For BGD, initial standard deviation was $\sigma = 0.06$, $\eta_\mu = 1$, and 2 MC samples were used. For FTML, SGD with learning rate $\lambda^{in} = 0.1$ was used in the inner loop, and Adam with learning rate 0.001 was used in the outer loop.

In split MNIST, the original image size 28x28 was used. Each task is a two-way classification. For W&H, $k = 5$ steps and a learning rate $\lambda^{in} = 0.1$ were used in the loop during training, initial standard deviation was $\sigma = 0.06$, $\eta_\mu = 1$, $\eta_\sigma = 1$, $J = 1$ MC sample was used. At testing time, no inner loop updates were used. For BGD, initial standard deviation was $\sigma = 0.06$, $\eta_\mu = 1$ and 2 MC samples were used. For FTML, SGD with learning rate $\lambda^{in} = 0.1$ was used in the inner loop, and Adam with learning rate 0.001 was used in the outer loop.

SINE CURVE REGRESSION The amplitudes and phases of sine curves were sampled uniformly from $[1.0, 5.0]$ and $[0, \pi]$ respectively. The learning rate for SGD was 0.01. For BGD, the initial standard deviation was 0.06, the learning rate for the mean was $\eta = 1$ and 2 Monte Carlo samples were used per iteration. For Reptile and FTML, we used SGD with learning rate 0.01 in the inner loop and Adam with learning rate 0.001 in the outer loop. For the What & How method, SGD with learning rate 0.01 was used in the inner loop and the initial standard deviation of the meta parameter distribution was 0.2, The learning rates for both the mean and standard deviation were chosen to be 1.0. The number of Monte Carlo samples was $J = 1$. For training, each mini-batch had 10 points, whose input values x were sampled uniformly from $[-5.0, 5.0]$. For testing, we used 50 equidistant points in $[-5.0, 5.0]$. Before the evaluation, each model was fine-tuned on 10 context data points. All methods except BGD were adapted for 500 SGD steps with a learning rate 0.01. For BGD, this learning rate often led to numerical instability, so we reduced the learning rate to 0.001 and applied 1000 steps of SGD before testing.

CONTINUAL GAN ON 2D GAUSSIAN MIXTURES The experimental setup was exactly the same as in [Metz et al., 2016]: the real distribution was a mixture of 8 Gaussians of standard deviation 0.02, and their means were equally spaced around a circle of radius 2. The generator and discriminator are both MLPs with 2 hidden layers of 128 relu units. In the vanilla GAN, the discriminator was optimized using Adam with a learning rate of 1e-4 and $\beta_1 = 0.5$. The WHGAN discriminator was optimized with an inner loop

Table 4.7: The architecture of the generator and discriminator of DCGAN. Here we use the following abbreviations: N= Number of filters, K= Kernel size, S= Stride size, P= Padding size. "Conv", "Dconv","BN" denote respectively the convolutional layer, transposed convolutional layer and batch normalization.

Layer	Generator	Discriminator
1	Dconv(256-K4-S1-P0), BN, Relu	Conv(N64-K4-S2-P1), Leaky-Relu
2	Dconv(N128-K4-S2-P1), BN, Relu	Conv(N128-K4-S2-P1), BN, Leaky-Relu
3	Dconv(N64-K4-S2-P1), BN, Relu	Conv(N256-K4-S2-P1), BN, Leaky-Relu
4	Dconv(N3-K4-S2-P1), Tanh	Conv(N1-K4-S1-P0), Sigmoid

learning rate 0.01, initial standard deviation 0.01, $\eta_\mu = 10$, $\eta_\sigma = 1$ and $J = 1$ MC sample. In both cases, the generator networks were optimized using Adam with a learning rate of 1e-3 and $\beta_1 = 0.5$.

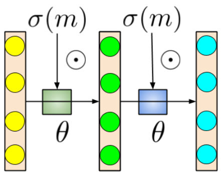
CONTINUAL GAN ON CIFAR10 For DCGAN, we used the original image size (32×32) of CIFAR10. The network architecture of DCGAN is summarized in Table 4.7. The noise vectors have dimension 128 and were drawn from standard normal distribution. Both networks were optimized using Adam with learning rate 0.0002, $\beta_1 = 0.5$ and $\beta_2 = 0.999$. Both real and fake mini-batches have size 64.

For WHGAN, we used exactly the same network architecture as the DCGAN baseline. The generator was also optimized using Adam with learning rate 0.0002, $\beta_1 = 0.5$ and $\beta_2 = 0.999$. For the discriminator, we used inner loop learning rate $\lambda^{in} = 0.01$, initial standard deviation $\sigma = 0.01$, $\eta_\mu = 100$, $\eta_\sigma = 1$, and MC number $J = 1$.

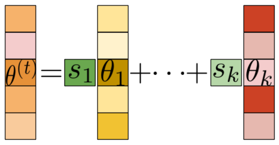
4.7.3 *Other Instantiations of the What & How Framework*

To show that the What & How framework is independent of the Reptile algorithm, here we introduce other variations of this framework using different meta-learning algorithms. The first instance is simply replacing Reptile with full MAML, to which we refer as MetaBGD in the following. In addition, we propose two other instances by adapting previous continual learning methods to this meta learning framework.

METACOG Context-dependent gating of sub-spaces [He and Jaeger, 2018], parameters [Mallya and Lazebnik, 2018] or units [Serra et al., 2018] of a single network has proven effective at alleviating catastrophic forgetting. Recently, Masse, Grant, and Freedman, 2018 showed that combining context dependent gating with a synaptic stabilization



(a) MetaCoG



(b) MetaELLA

Figure 4.4: Schematic diagrams of MetaCoG and MetaELLA.

method can achieve even better performance than using either method alone. Therefore, we explore the use of context dependent masks as our task representations, and define the task specific model as the sub-network selected by these masks.

At every time step t, we infer the latent masks m_t based on the context dataset $\mathcal{D}_t^{\text{cxt}}$ by one or a few steps of gradient descent of an inner loop loss function $\mathcal{L}^{\text{in}}$ with respect to m:

$$m_t := \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{cxt}}; \theta) = m^{\text{init}} - \lambda^{\text{in}} \cdot \nabla_m \mathcal{L}^{\text{in}}(\hat{f}(\cdot\,; \theta \odot \sigma(m)), \mathcal{D}_t^{\text{cxt}}), \tag{4.23}$$

where m^{init} is a fixed initial value of the mask variables, $\sigma(\cdot)$ is an element-wise sigmoid function to ensure that the masks are in $[0, 1]$, and $\odot$ is element-wise multiplication. In general, $\mathcal{L}^{\text{in}}$ can be any objective function. For instance, for a regression task, one can use a mean squared error with an L_1 regularization that enforces sparsity of $\sigma(m)$:

$$\mathcal{L}^{\text{in}}(\hat{f}(\cdot\,; \theta \odot \sigma(m)), \mathcal{D}_t^{\text{cxt}}) := \sum_{x_i, y_i \in \mathcal{D}_t^{\text{cxt}}} (\hat{f}(x_i; \theta \odot \sigma(m)) - y_i)^2 + \gamma \|\sigma(m)\|_1. \tag{4.24}$$

The resulting masks m_t are then used to gate the base network parameters θ_t in order to make a context-dependent prediction: $\hat{y}_t = \hat{f}(x_t; \theta_t \odot \sigma(m_t))$. Once the ground truth y_t is revealed, we can define the meta loss as the loss of the masked network on the current data: $\mathcal{L}^{\text{meta}}(\hat{f}(\cdot\,; \theta \odot \sigma(m_t)), \{(x_t, y_t)\})$ and optimize the distribution $q(\theta|\mu, \sigma)$ of task agnostic meta variable θ by online Bayesian inference.

The intuition here is that the parameters of the base network should allow fast adaptations of the masks m_t. Since the context-dependent gating mechanism is trained in a meta-learning fashion, we call this particular instance of our framework Meta Context-dependent Gating (MetaCoG). We note that while we draw our inspiration from the idea of selecting a subnetwork using the masks m_t, in the formulated algorithm m_t rather plays the role of modulating the parameters (i.e. in practice we noticed that entries of m_t do not necessarily converge to 0 or 1). Figure 4.4a shows a schematic diagram of MetaCoG.

In addition, the inner loop loss $\mathcal{L}^{\text{in}}$ used to infer the context variable m_t does not have to be the same as the meta loss $\mathcal{L}^{\text{meta}}$. In fact, one can choose an auxiliary loss function for $\mathcal{L}^{\text{in}}$ as long as it is informative about the current task.

METAELLA Another instance of the framework is based on the GO-MTL model [Kumar and au2, 2012] and the Efficient Lifelong Learning Algorithm (ELLA) [Ruvolo and Eaton, 2013]. In a multitask learning setting, ELLA tries to solve each task with a task specific parameter vector $\theta^{(t)}$ by linearly combining a shared dictionary of k latent model components $L \in \mathbb{R}^{d \times k}$ using a task-specific coefficient vector $s^{(t)} \in \mathbb{R}^k$: $\theta^{(t)} := Ls^{(t)}$, where L is learned by minimizing the objective function

$$\mathcal{L}^{\text{ella}}(L) = \frac{1}{T}\sum_{t=1}^{T} \min_{s^{(t)}} \left\{ \frac{1}{n^{(t)}} \sum_{i=1}^{n^{(t)}} \mathcal{L}\big(\hat{f}(x_i^{(t)}; Ls^{(t)}), y_i^{(t)}\big) + \mu \|s^{(t)}\|_1 \right\} + \lambda \|L\|_F^2. \quad (4.25)$$

Instead of directly optimizing $\mathcal{L}^{\text{ella}}(L)$, we adapt ELLA to the What & How framework by considering $s^{(t)}$ as the task representation returned by a What encoder and L as parameters of a How decoder. The objective $\mathcal{L}^{\text{ella}}$ can then be minimized in a continual meta learning fashion. At time t, current task representation s_t is obtained by minimizing the inner loop loss $\mathcal{L}^{\text{in}}(\hat{f}(\cdot; Ls), \mathcal{D}^{\text{cxt}}) := \frac{1}{|\mathcal{D}^{\text{cxt}}|}\sum_{x_i, y_i \in \mathcal{D}^{\text{cxt}}} \mathcal{L}(\hat{f}(x_i; Ls), y_i) + \mu\|s\|_1$ by one or a few steps of gradient descent from fixed initial value s^{init}: $s_t := \mathcal{F}^{\text{What}}(\mathcal{D}_t^{\text{cxt}}; L) = s^{\text{init}} - \lambda^{\text{in}} \cdot \nabla_s \mathcal{L}^{\text{in}}(\hat{f}(\cdot; Ls), \mathcal{D}_t^{\text{cxt}})$.

Similar to MetaCoG, the parametric distribution $q(L|\mu^L, \sigma^L) = \prod_i \mathcal{N}(L_i|\mu_i^L, \sigma_i^L)$ of the meta variable L can be optimized with respect to the meta loss $\mathcal{L}^{\text{meta}}(\hat{f}(\cdot; Ls_t), \{(x_t, y_t)\})$ using online Bayesian inference. Figure 4.4b shows a schematic diagram of MetaELLA.

4.7.4 *Experiments using other instantiations*

We compared BGD and Adam [Kingma and Ba, 2014] to the three instances introduced above: MetaBGD, MetaCoG and MetaElla. In all experiments, we presented N tasks consecutively and each task lasted for M iterations. At every iteration t, a batch of K samples $\mathcal{D}_t = \{x_{t,1}, \cdots x_{t,K}\}$ from the training set of the current task were presented to the learners, and the context data used for task inference was simply the previous mini-batch with their corresponding targets: $\mathcal{D}_t^{\text{cxt}} = \mathcal{D}_{t-1} \bigcup \{y_{t-1,1}, \cdots y_{t-1,K}\}$. At the end of the entire training process, we test the learners' performance on the testing set of every task, given a mini-batch of training data from that task as context data. Since the meta learners take five gradient steps in the inner loop for task inference, we also allowed BGD and Adam to take five gradient steps on the context data before testing their performances. In all experiments, the number of Monte Carlo samples was set to 10. In MetaCoG, the initial value of masks m_i^{init} was 0. In MetaELLA, we used

Table 4.8: Summary of hyperparameters used for the experiments described in the Appendix. λ^{in} are inner loop learning rates. σ_0 are initial values for standard deviation of the factorized Gaussian. η is the learning rate of the mean in BGD update rule. γ is the regularization strength for L1 norm of masks in MetaCoG. μ is the regularization strength for L1 norm of latent code in MetaELLA.

Hyperparameters	Sine Curve	Label-Permuted MNIST	Omniglot
MetaBGD λ^{in}	0.0419985	0.45	0.207496
σ_0	0.0368604	0.050	0.0341916
η	5.05646	1.0	15.8603
MetaCoG λ^{in}	0.849212	10.000	5.53639
σ_0	0.0426860	0.034	0.0133221
γ	1.48236e-6	1.000e-5	3.04741e-6
η	38.6049	1.0	80.0627
MetaElla λ^{in}	0.0938662	0.400	0.346027
σ_0	0.0298390	0.010	0.0194483
μ	0.0216156	0.010	0.0124128
η	42.6035	1.0	24.7476
BGD σ_0	0.0246160	0.060	0.0311284
η	20.3049	1.0	16.2192

$k = 10$ components in the dictionary, and the initial value of latent code s_i^{init} was set to $1/k = 0.1$. Adam baseline was trained with the default hyperparameters recommended in [Kingma and Ba, 2014]. The hyperparameters of other methods were tuned by a Bayesian optimization algorithm and were summarized in Table 4.8. Error bars for all experiments are standard deviations computed from 10 trials with different random seeds.

SINE CURVE REGRESSION In this experiment, we randomly generated 10 sine curves and present them sequentially to a 3-layer MLP. The amplitudes and phases of sine curves were sampled uniformly from $[1.0, 5.0]$ and $[0, \pi]$, respectively. For both training and testing, input points x were sampled uniformly from $[-5.0, 5.0]$. The size of training and testing sets for each task were 5000 and 100, respectively. Each sine curve was presented for 1000 iterations, and a mini-batch of 128 data points was provided at every iteration for training. The 3-layer MLP has 50 units with $\tanh(\cdot)$ non-linearity in each hidden layer. Figure 4.5 shows the mean squared error (MSE) of each task after the entire training process. Adam and BGD performed significantly worse than the meta learners, even though they had taken the same number of gradient steps on the context data. The reason for this large gap of performance becomes evident by looking at Figure 4.6, which shows the learners' predictions on testing data of the last task and the third task, given their corresponding context data. All learners could solve the last task almost perfectly, but when the context data of the third task was provided, meta learners could quickly remember it, while BGD and Adam were unable to adapt to the task they had previously learned.

LABEL-PERMUTED MNIST The experiment setup is the same as introduced in the chapter. Five tasks were created and presented sequentially. All tasks were presented for 1000 iterations and the mini-batch size was 128. The network is a MLP with 2 hidden layers of 300 ReLU units. As can be seen from Figure 4.7, all learners performed well on the last task. However, BGD and Adam had chance-level accuracy on previous tasks due to their incapability of inferring tasks from context data, while the meta learners were able to recall those tasks within 5 inner loop updates on the context data.

Figure 4.8 displays the accuracy curve when we played the tasks again, for 10 iterations each, after the first training process. The tasks were presented in the same order as they were learned for the first time. It is clear that after one iteration since the task changed, when the correct context data was available, the meta learners were able to recall the presented task to high accuracy, while Adam and BGD had to re-learn each task from scratch.

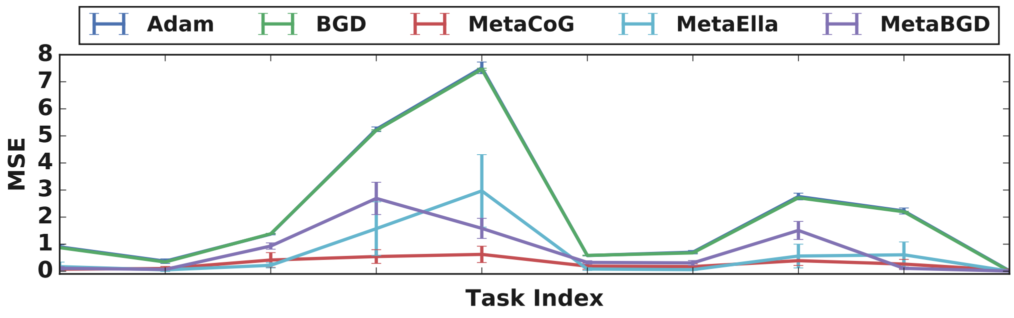


Figure 4.5: Testing loss per task at the end of the entire learning phase. Task 1 is the first seen task, and task 10 is the last. Lower MSE means better performance.

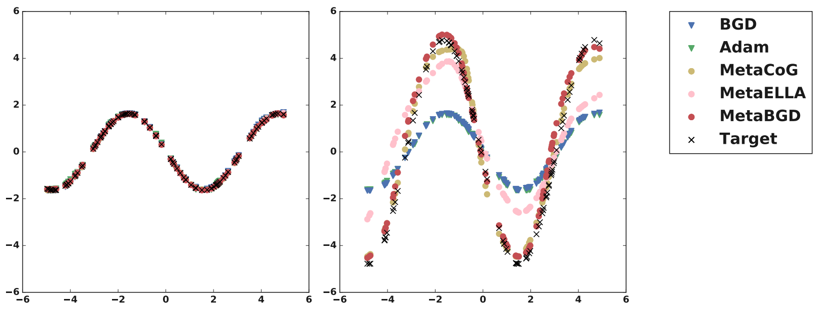


Figure 4.6: Predictions for the last task (left) and the third task (right) after the entire training process.

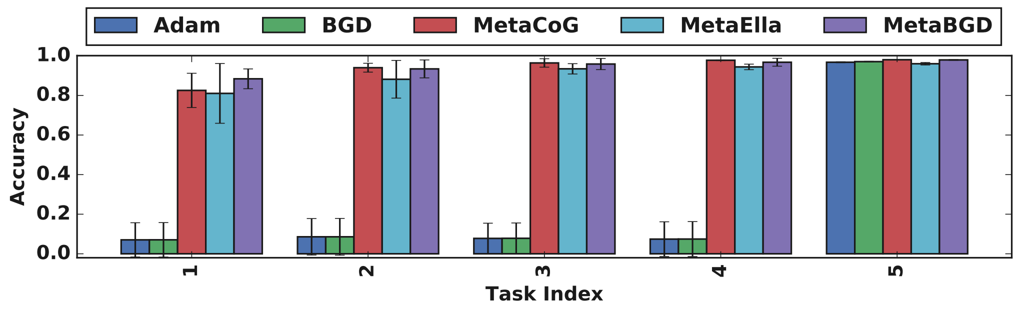


Figure 4.7: Testing accuracy of different tasks in the label-permuted MNIST experiment at the end of the entire training process.

OMNIGLOT In this experiment, we tested our framework and BGD by sequential learning of handwritten characters from the Omniglot dataset [Lake, Salakhutdinov, and Tenenbaum, 2015], which consists of 50 alphabets with various number of characters per alphabet. Considering every alphabet as a task, we presented 10 alphabets sequentially to a convolutional neural network and trained it to classify 20 characters from each alphabet. Out of the 20 images of each character, 15 were used for training and 5 for testing. Each alphabet was trained for 200 epochs with mini-batch size 128. The CNN used in this experiment has two convolutional layers, both with 40 channels and kernel size 5. ReLU and max pooling were applied after each convolution layer, and the output was passed to a fully connected layer of size 300 before the final layer.

Most continual learning methods (including BGD) require a multi-head output in order to overcome catastrophic forgetting in this set-up. The idea is to use a separate

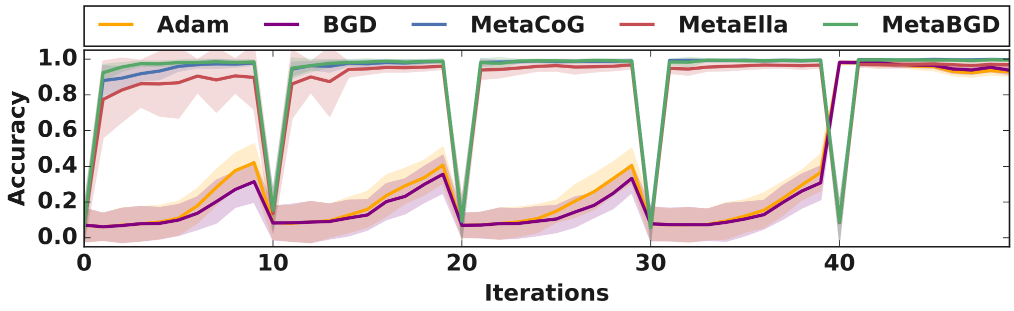


Figure 4.8: Accuracy curve when the label-permuted MNIST tasks are replayed for 10 iterations after the entire training process. The sudden drops of accuracy are due to task switching, when the context data are still from the previous task.

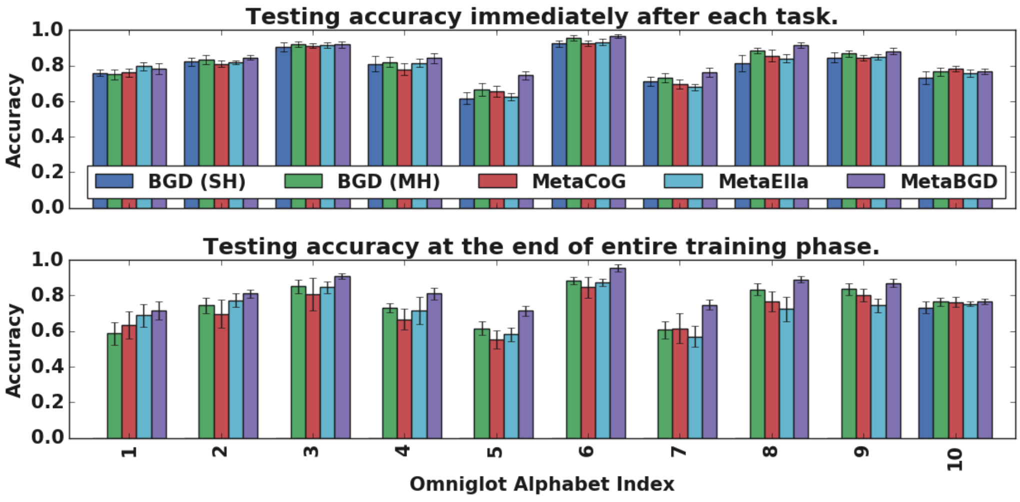


Figure 4.9: Testing accuracy of the sequential Omniglot task. BGD (MH) uses a multi-head output layer, whereas BGD (SH) and all meta learners use a single-head output layer. In the bottom plot, the accuracy of BGD(SH) are 0 for all tasks except the last one.

output layer per task, and to only compute the error on the current head during training and only make predictions from the current head during testing. Therefore, task index has to be available in this case in order to select the correct head. Unlike these previous works, we evaluated our framework with a single head of 200 output units in this experiment. Figure 4.9 summarizes the results of this experiment. For every task, we measured its corresponding testing accuracy twice: once immediately after that task was learned (no forgetting yet), and once after all ten tasks were learned. Our framework with a single head achieved comparable results as BGD with multiple heads, whereas BGD with a single head completely forgot previous tasks.

5 CONTINUAL LEARNING FROM THE PERSPECTIVE OF COMPRESSION

In this chapter, we study continual learning from the perspective of information theory and provide a formal definition of forgetting in terms of description lengths. Furthermore, we show that the prior-focused and likelihood-focused approaches to continual learning that we have introduced in Chapter 1 can be considered as approximations to two prequential coding methods in compression, namely, the Bayesian mixture code and maximum likelihood (ML) plug-in code. This offers an alternative view for unifying these two types of approaches. Finally, to address the limitations of the prior-focused approaches, we introduce a new continual learning paradigm by combining ML plug-in and Bayesian mixture codes. This chapter is a verbatim copy of [He and Lin, 2020][1] with additional preliminaries on minimum description length theory.

5.1 INTRODUCTION

Intuitively speaking, forgetting can be understood as loss of information about past data. Previous studies on this problem have indirectly measured forgetting by degradation of task-specific performance metrics such as classification accuracy [Lopez-Paz et al., 2017], regression error [Javed and White, 2019] or the Frechet Inception Distance (FID) [Lesort et al., 2019]. In order to explicitly study the problem of information loss, here we place continual learning within the framework of information theory and Minimum Description Length (MDL) [Grünwald, 2007].

According to the MDL principle, any regularity within a dataset can be used for compression, and learning these regularities using a model is equivalent to compressing the data with the model [Grünwald, 2007]. So the best model for a given dataset is the one that results in the minimum *total* description length of the dataset *together with* the model itself. Therefore, MDL formalizes the Occam's Razor principle for machine learning and provides a criterion for model selection. Other inspirations and foundations for MDL include the "comprehension is compression" hypothesis by Chaitin [2002] and the theory of inductive inference by Solomonoff [1964], the latter proved that the most likely explanation of some observations of the world is the smallest computer program, in terms of its Kolmogrov Complexity [Kolmogorov, 1968], that generates these observations, assuming the world is generated by an unknown computer.

1 This work was also presented as a spotlight talk at ICML2020 Workshop on Continual Learning.

The compression perspective has provided insights in various problems in machine learning, including generalization [Arora et al., 2018], unsupervised modeling [Ollivier, 2014] and information bottleneck [Tishby and Zaslavsky, 2015]. Recently, Blier and Ollivier [2018] showed that one particular coding method from the MDL toolbox called *prequential coding* outperforms the other coding methods at explaining why deep learning models generalize well despite their large number of parameters, displaying strong correlation between generalization and compression.

In this chapter, we show that the compression perspective can also shed light on continual learning. Our contributions are:

- We provide a formal definition of forgetting based on information theory, which allows us to evaluate continual learning methods for generative models in a principled way.
- We show that two major paradigms of continual learning (variational continual learning and generative replay) can be interpreted as approximations of two prequential coding methods (Bayesian mixture coding and ML plug-in coding) in the MDL framework, thus establishing a connection between continual learning and compression.
- We compare different CL methods for continual generative modeling using the proposed definition of forgetting and empirically study the limitations of the variational continual learning approaches.
- We introduce a new CL method that combines the prediction strategies of the two prequential coding methods and show that it improves over the variational continual learning.

5.2 MINIMUM DESCRIPTION LENGTH PRELIMINARIES

We start with a review of the basic concepts and results used in the MDL theory. A comprehensive introduction and overview of this field can be found in [Grünwald, 2007].

CODING In general, the word "coding" means to describe sequences of symbols from one alphabet by other sequences of symbols from some potentially different alphabet. Here, an alphabet is simply a finite or countable set, whose elements are called symbols. In the MDL setting, the sequence of symbols we want to describe is usually a data sequence $x^n := \{x_1, \ldots, x_n\}$ where each x_i is from a space of observation $\mathcal{X}$, which is countable or finite, and we always encode x^n with the binary alphabet $\mathbb{B} = \{0, 1\}$. In other words, we are mapping a sequence x^n to a binary sequence in the set $\mathbb{B}^* = \bigcup_{k \geqslant 1} \mathbb{B}^k$.

CODING SYSTEM Let $\mathcal{A}$ be an alphabet. A coding system C is a relation between $\mathcal{A}$ and $\mathbb{B}^*$. If $(a, b) \in C$, we say b is a *code word* for the *source symbol* a. A coding system is called *partial* if for some a, there is no b such that $(a, b) \in C$. A coding system is called *singular* or *lossy* if two different source symbols $a, a' \in \mathcal{A}, a \neq a'$ share the same code word (there exist b such that $(a, b) \in C$ and $(a', b) \in C$). In what follows, the alphabet $\mathcal{A}$ will in some cases represent the set of single outcomes ($\mathcal{A} = \mathcal{X}$). In other cases, it might represent the set of sequences of either a fixed length ($\mathcal{A} = \mathcal{X}^n$) or arbitrary lengths ($\mathcal{A} = \mathcal{X}^*$). Sometimes, it can also be a set of model parameters ($\mathcal{A} = \Theta$ for some parameter set Θ).

CODE A code is a non-singular coding system such that each $a \in \mathcal{A}$ is associated with at most one $b \in \mathbb{B}^*$. Therefore, a code may be identified with an encoding function $C : \mathcal{A} \to \mathbb{B}^* \cup \{\uparrow\}$ that maps each $a \in \mathcal{A}$ to its unique code word in $\mathbb{B}^*$ and $C(a) =\uparrow$ means a does not have a code word. In addition, since a code is non-singular, there is also a corresponding decoding function $C^{-1} : \mathbb{B}^* \to \mathcal{A} \cup \{\uparrow\}$ that maps a code word b to its unique source symbol $C^{-1}(b)$. $C^{-1}(b) =\uparrow$ means b is not a code word for any source symbol.

PREFIX CODE A code with the property that no code word is a prefix of any other code word is called a *prefix* code. With a prefix code, we know exactly when we are done decoding a code word if we decode from left to right, since we must be done when the sequence of symbols we see so far correspond to a code word. If not, we should proceed to the next symbol and check again. In MDL, we only work with non-singular (lossless) prefix coding systems, so in the rest of this chapter, the word "code" always means "prefix code".

CODE LENGTH FUNCTION Let C be some code for $\mathcal{A}$, then $\forall x \in \mathcal{A}$, the code length (or sometimes codelength) function $L_C(x)$ is the length (number of bits) of the code word (or description) of x when it is encoded with C. If C is a partial code and $C(x)$ is not defined, we set $L_C(x) = \infty$. In MDL, one is only interested in the code length function L_C but not the code C itself. If two codes C_1 and C_2 for the same alphabet use different code words but have the same code length, i.e. $\forall x \in \mathcal{A}, L_{C_1}(x) = L_{C_2}(x)$, then these two codes make no difference in the view of MDL. We use $\mathcal{L}_{\mathcal{A}}$ to denote the set of all functions L for which there exists a prefix code C such that $L(x) = L_C(x)$ for all $x \in \mathcal{A}$. Since the code length function is the only thing people are interested in MDL, "code length functions" are sometimes also referred to as "codes".

EXTENDED KRAFT INEQUALITY For any code C over countable alphabet $\mathcal{A}$, the code length function L_C must satisfy the inequality

$$\sum_{x \in \mathcal{A}} 2^{-L_C(x)} \leqslant 1. \tag{5.1}$$

Conversely, given a set of code lengths $\{L_x | x \in \mathcal{A}\}$ that satisfy this inequality, there exists a prefix code C with these code word lengths: $L_C(x) = L_x, \forall x \in \mathcal{A}$. A proof of the extended Kraft inequality can be found in Cover and Thomas [2006].

SHANNON-FANO CODE Let P be a probability distribution over the countable space $\mathcal{X}$, it follows from Kraft inequality that there exists a code C for $\mathcal{X}$ such that $\forall x \in \mathcal{X}$: $L_C(x) = \lceil -\log P(x) \rceil$. This is called the Shannon-Fano code.

NON-INTEGER CODE LENGTHS Since a code length refers to the number of bits of a code word, its value should be integer (or ∞ if the code word is not defined). However, in MDL, we use code length only as a metric to measure the amount of information and do not care about the actual encodings, hence we adopt the idealized view of code lengths which does not require them to be integers. As shown in Grünwald [2007], this noninteger view of code lengths is not only harmless in practice but also more fundamental: it makes the code lengths invariant to the size of encoding alphabet; it offers an interpretation of description methods as gambling schemes described in Kelly gambling; it also makes the mathematics easier and more elegant.

Once we dropped the requirement of integer values, the only requirement we have for code length functions is the Kraft inequality, and this leads to the following new definition of codelength functions:

GENERALIZED CODELENGTH FUNCTION The set of all codelength functions for sample space $\mathcal{X}$ is redefined in MDL as

$$\mathcal{L}_{\mathcal{X}} = \{L : \mathcal{X} \to [0, \infty] | \sum_{x \in \mathcal{X}} 2^{-L(x)} \leqslant 1\}. \tag{5.2}$$

Notably, this generalized definition applies to not only countable but also continuous sample spaces $\mathcal{X}$. In the latter case, the sum in the Kraft inequality above should be replaced by an integral $\int_{x \in \mathcal{X}} 2^{-L(x)} dx \leqslant 1$. This new definition of codelength functions also allows us to extend the definition of Shannon-Fano code to probability density functions.

SHANNON-FANO CODE FOR CONTINUOUS SPACE Let p be a probability density function over a continuous space $\mathcal{X}$. It's not hard to see that $L(x) := -\log p(x)$ satisfies

$\int_{x \in \mathcal{X}} 2^{-L(x)} dx \leqslant 1$, hence it is a generalized codelength function, which is also called the Shannon-Fano code[2].

TWO-PART CODE MDL PRINCIPLE Given a dataset $x^n := \{x_1, \ldots, x_n\}$ where each x_i is from a space of observation $\mathcal{X}$, the MDL principle tells us to choose the model θ that minimizes the overall description length of x^n, which can be divided into two parts:

$$L(x^n) = L(x^n|\theta) + L(\theta). \tag{5.3}$$

The first part $L(x^n|\theta)$ corresponds to the code lengths of x^n when encoded with the help of a model parametrized by θ, and the second part $L(\theta)$ corresponds to the codelength of the model itself. A model with more capacity can fit the data better, thus resulting in shorter codelength of the first part. However, higher capacity might also lead to longer description length of the model itself. Therefore, the best explanation for x^n, according to MDL, is given by the model that minimizes the total length $L(x^n)$.

To understand coding schemes in MDL intuitively, it is usually helpful to imagine the scenario where Alice wants to send a dataset x^n to Bob as efficiently as possible. What the two-part code says is that Alice can send the dataset compressed by some model to minimize the communication cost. However, the descriptions of the model itself have to be transmitted as well. Therefore, the total communication cost includes both the transmission of the data encodings as well as the transmission of the model parameters. Note that, before the parameters are transmitted, we assume that Alice and Bob have already agreed on a model class (such as a Gaussian distribution or "a neural network with a hidden layer of 200 neurons"). If not, they could also first communicate this meta information and since the amount of meta information to send does not depend on the data, its communication cost is negligible in the long run.

Given the model parameters θ and the data x^n, we can compute the overall description length using Equ. 5.3. For probabilistic models, the natural choice for computing the first part $L(x^n|\theta)$ is the Shannon-Fano code $L(x^n|\theta) = -\log p(x^n|\theta)$. In order to compute the second part $L(\theta)$, we also need an encoding of the parameters θ. One strategy to encode the parameters is to consider them also as random variables and use the (variational) Bayesian coding scheme.

2 There is no problem using this continuous version of Shannon-Fano code in MDL, since we only need to measure and compare the amount of information. However, in communication and compression applications, it is impossible to encode samples of continuous space such as real numbers of infinite precision because an infinite amount of information is needed to represent them. In those cases, one has to first quantize the real numbers into intervals, then assign a probability mass to each interval based on the underlying probability density function [Hinton and Van Camp, 1993].

BAYESIAN AND VARIATIONAL CODES For any parametric model family $p(\cdot|\theta), \theta \in \Theta$, if we consider the parameters θ as random variables, then any prior $p(\theta)$ over Θ corresponds to a Bayesian code:

$$L^{\text{Bayes}}(x^n) := -\log p(x^n) = -\log \int_\theta p(x^n|\theta)p(\theta)d\theta. \tag{5.4}$$

However, the integral in 5.4 is rarely tractable, so we usually use variational methods to compute the variational code. Given the model family $p(\cdot|\theta), \theta \in \Theta$ and a prior $p(\theta)$, for any distribution $q(\theta)$ over Θ, there exists an encoding of x^n called the variational code with the the following codelength:

$$L_q^{\text{Var}}(x^n) := \mathbb{E}_{q(\theta)}\left[-\log(x^n|\theta)\right] + D_{\text{KL}}\left[q(\theta)\|p(\theta)\right]. \tag{5.5}$$

It is easy to see that 5.5 is just the additive inverse of the evidence lower bound (ELBO) of $\log p(x^n)$, thus L_q^{Var} is an upper bound for the Bayesian code L^{Bayes}: for any q, we have

$$L_q^{\text{Var}}(x^n) \geqslant L^{\text{Bayes}}(x^n), \tag{5.6}$$

where equality holds only when q equals the true Bayesian posterior $p(\theta|x^n) = p(x^n|\theta)p(\theta)/p(x^n)$.

The first term in (5.5) corresponds to $L(x^n|\theta)$, describing the expected code length of x^n if encoded with θ sampled from the approximate posterior $q(\theta)$. The second term in (5.5) corresponds to $L(\theta)$, which is the average code length of θ using the "bits-back" argument given by Hinton and Van Camp [1993]. The "bits-back" coding describes the following coding scheme:

Alice and Bob first agree on the parametric model family $p(x^n|\theta)$, a prior distribution $p(\theta)$, and an algorithm for learning variational posterior distributions from data. To communicate the data x^n, Alice runs the learning algorithm on x^n to obtain a variational posterior $q(\theta)$ and sample a random parameter θ from $q(\theta)$ using a source of random bits. Alice then sends the random sample θ encoded by the prior distribution $p(\theta)$, which incurs a communication cost[3] of $L_p := -\log p(\theta)$. Finally, Alice encodes x^n using θ and sends the encodings to Bob with a communication cost of $-\log p(x^n|\theta)$.

On the receiving end, Bob first decodes θ from the first message using $p(\theta)$. He can then decode x^n from the second message using $p(x^n|\theta)$. Once he has the data, he can run the same algorithm Alice used to recover the exact same variational posterior

3 As explained by Hinton and Van Camp [1993], the actual communication cost for sending the parameters should also include a term that depends on the precision for quantization. However, here we ignore this additional term since it is independent of the data and the model, and it will anyway be canceled out when we get the bits back.

$q(\theta)$. Now, since Bob knows the posterior of parameters and he knows which random parameters were sent to him, he can also recover the random bits that Alice used to sample θ. Therefore, these random bits are also transmitted to Bob successfully and they should be subtracted from the overall communication cost to get the net cost of sending the model θ and the data x^n. The number of random bits used to collapse $q(\theta)$ to one sample θ is $L_q := -\log q(\theta)$. Thus we can get the variational code in 5.5 by taking the expected value of the net overall communication cost $-\log p(x^n) + L_p - L_q$ over the distribution $q(\theta)$. In particular, the net cost for communicating the parameters is

$$L^{\mathrm{Var}}(\theta) = \mathbb{E}_{q(\theta)}\left[L_p - L_q\right] = D_{\mathrm{KL}}\left[q(\theta)\|p(\theta)\right]. \tag{5.7}$$

PREQUENTIAL (ONLINE) CODE There is also a way to compute the total description length $L(x^n)$ without explicitly separating it into two parts, which is to use the prequential or online code, based on the prequential approach to statistics [Dawid, 1984]. The prequential coding considers a dataset x^n as a sequence of observations and encode each observation x_t at time t by predicting its value based on past observations x^{t-1}. Note that for any x^t, we can write its Shannon-Fano code as a sum of negative log conditional probabilities:

$$-\log p(x^n) = \sum_{t=1}^{n} -\log p(x_t|x^{t-1}). \tag{5.8}$$

Hence the total description length can be computed by accumulating the code length $-\log p(x_t|x^{t-1})$ at every step t using a prediction strategy. Formally, a prediction strategy is a function $S : \bigcup_{1\leqslant t\leqslant n} \mathcal{X}^{t-1} \to \mathcal{P}_{\mathcal{X}}$ that maps any initial observations x^{t-1} to a distribution on $\mathcal{X}$ corresponding to the conditional model $p(X_t|x^{t-1})$. For example, the *Bayesian mixture code* uses the following prediction strategy:

$$p(x_t|x^{t-1}) = \int_\theta p(x_t|\theta)p(\theta|x^{t-1}), \tag{5.9}$$

where $p(\theta|x^{t-1})$ is the posterior of model parameters given all previous data x^{t-1}. Another example of prequential coding is the *maximum likelihood (ML) plug-in code*, which encodes each observation by

$$p(x_t|x^{t-1}) = p(x_t|\theta(x^{t-1})), \tag{5.10}$$

where $\theta(x^{t-1})$ is given by a maximum likelihood estimator:

$$\theta(x^{t-1}) = \arg\max_\theta \log p(x^{t-1}|\theta). \tag{5.11}$$

The communication scenario corresponding to the prequential coding is as follows: Before sending any messages, Alice and Bob agrees on an initial model $p(x)$ and a prediction strategy S. Then Alice encodes the first data point x_1 using the initial distribution $p(x)$ and sends it to Bob with a communication cost of $-\log p(x_1)$. Once x_1 is sent, Alice can update the model by S to get $p(x|x_1)$, which is used to transmit x_2 at the cost of $-\log p(x_2|x_1)$. On Bob's side, upon receiving the code for x_1, Bob decodes the message using $p(x)$ and runs the prediction strategy to get the same $p(x|x_1)$ for decoding x_2. As such, they can repeat this process to transmit x_3, x_4 and so on.

It is worth noting that the prequential approach does not require the sequence of data x^n to be i.i.d. For instance, Alice and Bob can also train a recurrent neural network (RNN) as the prediction strategy to compute $p(x_t|x^{t-1}) = p(x_t|h_{t-1}, \theta(x^{t-1}))$. In this case, they need to update both the RNN weights $\theta(x^{t-1})$ and the hidden state h_{t-1} at every step.

In practice, the conditional model $p(x|x^{t-1})$ might not be updated at every t but only at certain time steps $0 < t_1 < t_2 < \cdots < t_m \leqslant n$, and all data between two updates are encoded with the most recent model. If the prediction strategy relies on some stochastic process such as sampling or stochastic gradient descent, Alice and Bob also need to agree on a random seed before they start the transmission process so that they can obtain the same conditional probabilities at each step.

As we will show in the next section, this online incremental view of prequential coding is well-aligned with the setting of continual learning.

5.3 CONTINUAL LEARNING

Continual learning studies the scenario where data are provided at different times in a sequential manner, and the learning algorithm incrementally updates the model parameters as more data are presented. However, due to reasons such as privacy or limited memory budget, the learning algorithm may not always have access to all the data x^{t-1} it has seen so far. Alternatively, even if all data are stored, naively retraining on all previous data will increase the computational complexity per step over time, which may not be affordable for very long data streams. On the other hand, if the model is only trained on the data presented recently, as we have seen in previous chapters, it usually suffers from "catastrophic forgetting". This problem is especially severe when the data stream is non-stationary. Here we provide a definition of forgetting that allows us to precisely measure the amount of information forgotten.

5.3.1 *An Information-Theoretic Definition of Forgetting*

We can see from (5.3) that when the data x^n is compressed with the help of a model, the information of x^n is divided into two parts, with one part contained in the code words of x^n and the other part contained in the model. Hence, the model alone is not enough to recover the data losslessly, it requires also the code words of the compressed data, which can be understood as the information of x^n not captured by the model. This intuition leads to our definition of forgetting: for any data point x, the amount of forgetting about x caused by changing the model parameters from θ_i to θ_j is given as the increase of the codelength of x:

$$\mathcal{F}[\theta_j/\theta_i](x) := \max(L(x|\theta_j) - L(x|\theta_i), 0). \tag{5.12}$$

In other words, the number of extra bits required for the model to losslessly recover x is the amount of information the model has forgotten about x. The $\max(\cdot, 0)$ function in (5.12) is used to prevent "negative forgetting": if the codelength does not increase, there is no forgetting[4].
In the prequential setting, at any time t, we can measure the *cumulative average forgetting* of a learning algorithm over the entire sequence x^t seen so far by the following metric:

$$\mathcal{C}(x^t) := \frac{1}{t}\sum_{i=1}^{t} \mathcal{F}[\theta_t/\theta_i](x_i) = \frac{1}{t}\sum_{i=1}^{t} \max(L(x_i|\theta_t) - L(x_i|\theta_i), 0), \tag{5.13}$$

where θ_i are the parameters given by the learning algorithm at time i, after seeing x^i.

5.3.2 *Prequential Interpretation of CL Approaches*

A primary goal of continual learning is to overcome or to alleviate catastrophic forgetting. Many CL methods have been proposed for this purpose [Parisi et al., 2019]. Some common paradigms emerged from these methods. One paradigm is based on the sequential nature of Bayesian inference and it applies variational methods to approximating the parameter posterior. The approximated posterior is then used as the prior for future learning. Examples of this paradigm include Bayesian Online Learning [Opper, 1998], Variational Continual Learning (VCL) [Nguyen et al., 2018] and Online Structured Laplace Approximations [Ritter, Botev, and Barber, 2018]. Some regularization-based CL methods such as Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017] and

4 Mathematically, "negative forgetting" is possible and it corresponds to the concept of backward transfer [Lopez-Paz et al., 2017] in continual learning. But here we want to be aligned with the the conventional understanding of forgetting as information loss, and refer to "negative forgetting" as backward transfer when it happens.

Uncertainty-guided Continual Bayesian Neural Networks (UCB) [Ebrahimi et al., 2020] are also motivated by this idea. Another paradigm combats forgetting by training a generative model of past data. When new data are provided later, the samples from the generative model are replayed instead of the original data. Representative works of this class are Deep Generative Replay (DGR) [Shin et al., 2017], Memory Replay GAN [Wu et al., 2018] and Continual Unsupervised Representation Learning (CURL) [Rao et al., 2019]. Farquhar and Gal [2019a] proposed a unifying Bayesian view that encompasses both paradigms and named them prior-focused and likelihood-focused approaches, respectively. From the perspective of MDL, these two types of approaches can be seen as approximations of the two prequential coding methods introduced before. We discuss their relationships below using VCL[5] and Generative Replay as examples.

VCL The first paradigm corresponds to the Bayesian mixture code, which uses (5.9) as its prediction strategy. Since the exact posterior $p(\theta|x^{t-1})$ in (5.9) depends on all previous data x^{t-1}, which might not always be available in the continual learning scenario, VCL therefore uses a variational model $q(\theta|x^{t-1})$ to approximate the real posterior $p(\theta|x^{t-1})$. By the Bayes' rule, we have

$$p(\theta|x^t) = \frac{p(x_t|\theta)p(\theta|x^{t-1})}{Z_t}, \tag{5.14}$$

where $Z_t := \int_\theta p(x_t|\theta)p(\theta|x^{t-1})$ is a normalizing factor that does not depend on θ. Therefore, $q(\theta|x^t)$ can be updated recursively by minimizing the following KL divergence at every step t:

$$q(\theta|x^t) = \arg\min_{q(\theta)} D_{KL}(q(\theta)||\frac{1}{Z_t}p(x_t|\theta)q(\theta|x^{t-1})). \tag{5.15}$$

The first approximate posterior $q(\theta|x^0)$ is usually a fixed prior chosen before learning starts: $q(\theta|x^0) = p(\theta)$. Since Z_t does not depend on θ, it is not required for the optimization above.

REPLAY The second paradigm, on the other hand, corresponds to the ML plug-in code, which uses the prediction strategy defined in (5.10) and (5.11). The objective function in (5.11) requires all previous data x^{t-1}. In order to relax this requirement for

5 Here we use the term "VCL" to refer to the online variational Bayesian method used in [Nguyen et al., 2018]. The original VCL paper also combines it with a coreset to improve its performance. However, since we want to compare the differences between prior-focused and likelihood-focused approaches and coreset is a form of likelihood-focused approach, we exclude the usage of coreset for VCL here.

continual learning, replay-based methods approximate the real objective function using a generative model $p(x|\theta)$. Assuming $x_i \in x^t$ are independent given θ, we have:

$$\begin{aligned}\log p(x^t|\theta) &= \sum_{i=1}^{t} \log p(x_i|\theta) = \log p(x_t|\theta) + \sum_{i=1}^{t-1} \log p(x_i|\theta) \\ &\approx \log p(x_t|\theta) + (t-1)\mathbb{E}_{p(x|\theta_{t-1}))}[\log p(x|\theta)] \\ &= \log p(x_t|\theta) + (t-1)\int_x p(x|\theta_{t-1}) \log p(x|\theta), \end{aligned} \tag{5.16}$$

where θ_{t-1} are the parameters of the generative model at time $t-1$ and the expected value can be computed by Monte Carlo sampling. In this way the generative model can be recursively updated by the following rule, based on the maximum likelihood principle:

$$\theta_t = \arg\max_{\theta} \log p(x_t|\theta) + (t-1)\int_x p(x|\theta_{t-1}) \log p(x|\theta). \tag{5.17}$$

MEASURING FORGETTING AND COMPRESSION We empirically compare the two prequential coding methods and their continual learning approximations in terms of both forgetting and compression on the MNIST dataset. Data are presented in a class-incremental fashion: first only images of digit zero, then only images of digit one, and so on. Throughout the paper, we use a variational autoencoder (VAE) [Kingma and Welling, 2013] for encoding and generative modelling. In this experiment, we update the model and measure the performance after presenting an entire class of images.

Fig.5.1 shows the cumulative average forgetting after learning each digit. Since ML plug-in and Bayesian Mixture methods are trained on all data seen so far, they have almost zero forgetting. VCL and Replay, on the other hand, do not have access to previous data, so information are lost over time. However, thanks to the variational posterior and generated samples, they achieve less forgetting than the Catastrophic baseline, which only trains the model on images of the current digit.

The results on prequential coding as defined in (5.8) are shown in Fig. 5.2. After all the images of one class are presented, the model is updated and used to encode all the images of the next class. The subplot (a) shows the codelength of each class, since models are only trained on the images from previous classes, the results shown here correspond to the concept of forward transfer in continual learning. We can see that Bayesian Mixture and VCL have high codelength at the beginning, when there is no forgetting yet. The reason for this phenomenon is discussed in Sec. 5.4.1. The subplot (b) shows the prequential codelength of the entire MNIST dataset measured in bits per dimension (bpd) using different methods as the prediction strategy.

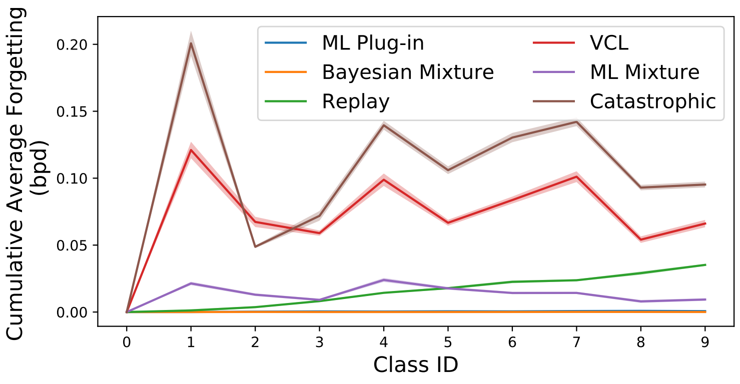


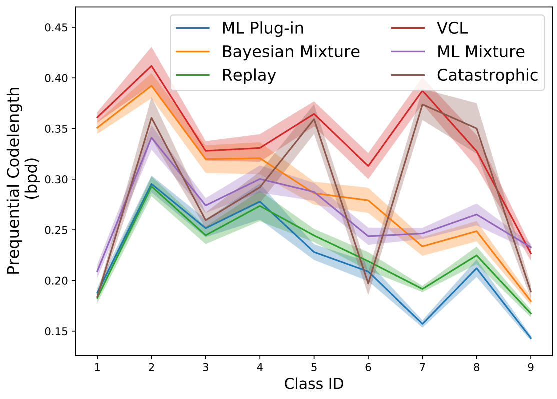
Figure 5.1: Cumulative average forgetting (defined in (5.13)) measured after learning each digit in MNIST. The resulting codelength was divided by the number of pixels in an image to get bits per dimension (bpd). The means and standard errors of the mean (SEM) were computed over ten random mini-batches of 32 images from each class. The shaded region correspond to 3 times of SEM. Here "Catastrophic" corresponds to the method that trains the model only on images from the current class. The "ML Mixture" method is introduced in Sec. 5.5. Both ML plug-in and Bayesian Mixture have near zero forgetting, so their curves are overlapping with each other.

5.4 LIMITATIONS OF VCL

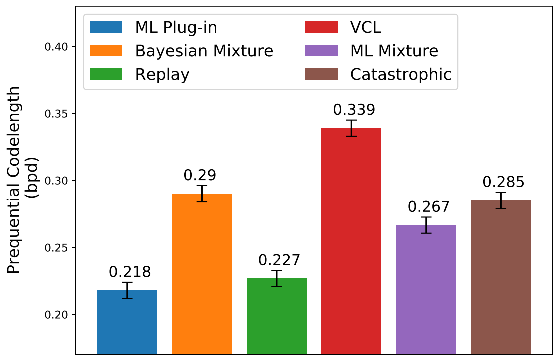
It can be seen from both Fig.5.1 and 5.2, VCL performs poorly compared to generative replay. A systematic comparison between VCL and replay on discriminative tasks was carried out by Farquhar and Gal [2019b]. VCL was reported to work similarly to generative replay on simple tasks like Permuted MNIST. On the more challenging Split MNIST task, VCL requires both a multi-head output layer and a coreset of past examples in order to be comparable with generative replay. For generative modeling, the results in Fig. 5.1 confirm the limitation of VCL at retaining information of past data. In this section, we analyze the reasons that limit the performance of VCL.

5.4.1 *Overhead of Bayesian Mixture due to Prior*

With deep generative models, the integration in Bayesian mixture code defined in (5.9) is intractable because $p(x|\theta)$ is a neural network generator. Therefore, the ELBO is used in practice as an approximation, which adapts $q(\theta)$ in (5.5) to minimize the total description length. We make two observations here. First, in VCL, the best achievable description



(a)



(b)

Figure 5.2: Compression performance of different coding methods on the MNIST dataset. The shaded region correspond to 3 times of SEM. **(a)** Prequential codelength of images in class i, measured after the model has seen all images in class $i-1$. Since the images of the first class are encoded with a predefined model for all methods, their codelength does not affect the comparison results. **(b)** The total prequential codelength of the MNIST dataset, averaged over number of samples.

length, corresponding to L^{Bayes} in (5.6), is predetermined by the selected prior $p(\theta)$. Second, compared to ML plug-in code, which directly minimizes the encoding length $L(x|\theta)$, the approximate Bayesian code has an extra KL regularization, indicating a longer description length $L(x|\theta)$ for the code words. Since we do not explicitly encode data and keep their code words in continual learning, a larger encoding length means more information is lost about the data as the online process continues.

This overhead is negligible asymptotically if the data sequence is very long. This can be seen from the relative weights of the two terms in (5.5). The relative weight of the KL regularization diminishes when the data sequence grows longer. This is why we see a high initial codelength for Bayesian Mixture in Fig. 5.2 (a), but gradually it closes the gap to ML plug-in.

However, for a sequence with fixed length, the overhead due to prior can be significant and is very sensitive to the choice of the prior. Figure 5.3 compares the prequential codelength of Bayesian Mixture code using Gaussion priors with different standard deviation σ. One can see that good performance can be achieved only when σ is within a small region around 0.1. And the best description length is obtained when the prior is not used. Similar effects were observed for VCL in [Swaroop et al., 2019], where the authors showed the choice of prior variance has great impact on the performance of VCL.

5.4.2 *The Variational Gap*

As we describe before, in continual learning, the true posterior $p(\theta|x^{t-1})$ is not available, thus VCL approximates it with a variational posterior $q(\theta|x^{t-1})$. The underlying assump-

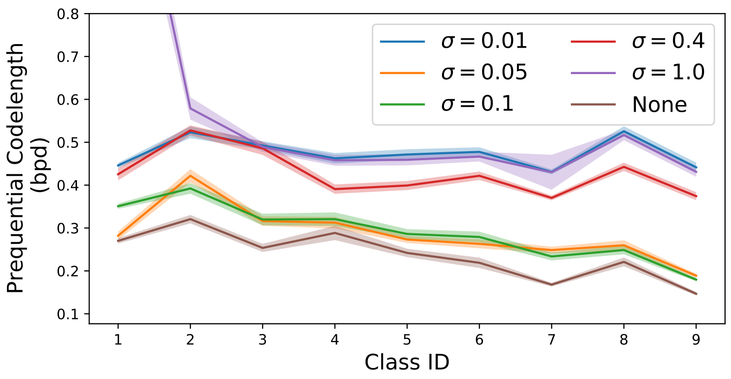


Figure 5.3: Prequential description length of Bayesian Mixture code using Gaussian priors with different standard deviations σ. Here, "None" means no prior is used, and the KL term in (5.5) is not optimized.

tion is that when their KL divergence is small enough: $D_{KL}[q(\theta|x^{t-1})\|p(\theta|x^{t-1})] < \epsilon$, the following approximations are good enough for continual learning:

$$q(\theta|x^{t-1}) \approx p(\theta|x^{t-1}) \tag{5.18}$$

$$\mathcal{L}_t^{VCL}(q) = \int_\theta q(\theta|x^t)\log p(X_t|\theta) - D_{KL}[q(\theta|x^t)\|p(\theta|x^{t-1})] \tag{5.19}$$

$$\approx \int_\theta q(\theta|x^t)\log p(X_t|\theta) - D_{KL}[q(\theta|x^t)\|q(\theta|x^{t-1})]. \tag{5.20}$$

However, for VCL with a simple distribution family such as diagonal Gaussian, this assumption can hardly hold, as the real posterior is proportional to $p(x^{t-1}|\theta)p(\theta)$, which is usually complex and multimodal. Since the only difference between VCL and Bayesian Mixture code is the substitution of (5.19) by (5.20), the error between these two terms is entirely responsible for their large performance gap observed in Fig.5.1 and 5.2. One potential remedy to this problem is to use more complex distributions on the parameter space, for example, normalizing flow as a Bayesian hypernetwork [Krueger et al., 2017]. However, unlike Gaussian distributions, the KL divergence of complex distributions usually cannot be calculated analytically. It is also unclear whether sampling in the parameter space to minimize $D_{KL}[q(\theta|x^{t-1})\|p(\theta|x^{t-1})]$ would be any cheaper than sampling in the data space for replay.

Furthermore, the approximation error between (5.19) and (5.20) could be arbitrarily large even if $D_{KL}[q(\theta|x^{t-1})\|p(\theta|x^{t-1})] < \epsilon$, which only states that $\log\frac{q(\theta|x^{t-1})}{p(\theta|x^{t-1})}$ is small in its expectation over the distribution $q(\theta|x^{t-1})$. As shown in (5.21), the approximation error is the expectation of the same log ratio over $q(\theta|x^{t-1})p(x_t|\theta)/Z_t$. Unfortunately, this distribution could be very different from $q(\theta|x^{t-1})$. It depends on $p(x_t|\theta)$, which is not available at the time $q(\theta|x^{t-1})$ is optimized. One sufficient condition to bound the substitution error is $\max_\theta|\log\frac{q(\theta|x^{t-1})}{p(\theta|x^{t-1})}| < \epsilon$. However, this involves a minimax optimization of the log ratio, which is a hard problem in itself.

$$\Delta(\mathcal{L}_t^{VCL}) = -\int_\theta q(\theta|x^t)\log\frac{q(\theta|x^{t-1})}{p(\theta|x^{t-1})} = -\int_\theta \frac{1}{Z_t}q(\theta|x^{t-1})p(x_t|\theta)\log\frac{q(\theta|x^{t-1})}{p(\theta|x^{t-1})}. \tag{5.21}$$

5.5 ML MIXTURE CODE

In order to overcome the two limitations discussed above, we can combine the prediction strategies of Bayesian mixture and ML plug-in using a hierarchical model with hyperparameters ϕ. The resulting prediction strategy, which we call *maximum likelihood (ML) mixture*, is as follows:

$$p(x_t|x^{t-1}) = p(x_t|\phi(x^{t-1})) = \int_\theta p(x_t, \theta|\phi(x^{t-1})) = \int_\theta p(X_t|\theta)p(\theta|\phi(x^{t-1})), \tag{5.22}$$

where $p(x_t|\theta)$ is a generative model of x_t parameterized by θ (for instance, a variational auto-encoder with weights θ) and $p(\theta|\phi)$ is a density function over parameters given hyper-parameters ϕ (for instance, if $p(x_t|\theta)$ is a Bayesian neural network whose weights are drawn from a diagonal Gaussian distribution, then $p(\theta|\phi)$ would be a Gaussian and ϕ are the mean and standard deviation of θ. $\phi(x^t)$ is defined as the hyper-parameters that maximize $\log p(x^t|\phi)$:

$$\phi(x^t) = \arg\max_\phi \log p(x^t|\phi) = \arg\max_\phi \sum_{i=1}^{t} \log \int_\theta p(x_i|\theta)p(\theta|\phi). \tag{5.23}$$

Hence it is still a maximum likelihood approach but on the hyper-parameter level. Since the integral above is intractable, in practice, we instead maximize its lower bound $\mathcal{L}_t^{\mathrm{MLM}}(\phi)$:

$$\begin{aligned}\log p(x^t|\phi) &= \sum_{i=1}^{t} \log \int_\theta p(x_i|\theta)p(\theta|\phi) \\ &\geqslant \sum_{i=1}^{t} \int_\theta q(\theta|x_i) \log \frac{p(x_i|\theta)p(\theta|\phi)}{q(\theta|x_i)} =: \mathcal{L}_t^{\mathrm{MLM}}(\phi).\end{aligned} \tag{5.24}$$

Computing $\mathcal{L}_t^{\mathrm{MLM}}(\phi)$ requires all past data x^t, for continual learning, in order to remove any dependency on previous data, we can rewrite $\mathcal{L}_t^{\mathrm{MLM}}(\phi)$ recursively with the following trick:

$$\begin{aligned}
\mathcal{L}_t^{\mathrm{MLM}}(\phi) &= \sum_{i=1}^{t} \int_\theta q(\theta|x_i) \log \frac{p(x_i|\theta)p(\theta|\phi)}{q(\theta|x_i)} && (5.25)\\
&= \int_\theta q(\theta|x_t) \log \frac{p(x_t|\theta)p(\theta|\phi)}{q(\theta|x_t)} + \sum_{i=1}^{t-1} \int_\theta q(\theta|x_i) \log \frac{p(x_i|\theta)p(\theta|\phi)}{q(\theta|x_i)} && (5.26)\\
&= \int_\theta q(\theta|x_t) \log \frac{p(x_t|\theta)p(\theta|\phi)}{q(\theta|x_t)} + \sum_{i=1}^{t-1} \int_\theta q(\theta|x_i) \log \frac{p(x_i|\theta)p(\theta|\phi_{t-1})}{q(\theta|x_i)}\\
&+ \sum_{i=1}^{t-1} \int_\theta q(\theta|x_i) \log \frac{p(\theta|\phi)}{p(\theta|\phi_{t-1})} && (5.27)\\
&= \int_\theta q(\theta|x_t) \log \frac{p(x_t|\theta)p(\theta|\phi)}{q(\theta|x_t)} + \mathcal{L}_{t-1}^{\mathrm{MLM}}(\phi_{t-1})\\
&+ \sum_{i=1}^{t-1} \int_\theta q(\theta|x_i) \log \frac{p(\theta|\phi)}{p(\theta|\phi_{t-1})}. && (5.28)
\end{aligned}$$

This result motivates us to use a greedy continual learning update rule, for which we assume that, at time t, the second term in (5.28) $\mathcal{L}_{t-1}^{\mathrm{MLM}}(\phi_{t-1})$ has already been optimized in the previous step and hence we only optimize the first and the last terms in (5.28).

Furthermore, we know from Hoffman and Johnson [2016] that if $p(\theta|\phi_{t-1})$ can be any arbitrary probability density function, then the optimal solution for $p(\theta|\phi_{t-1})$ that maximizes $\mathcal{L}_{t-1}^{\mathrm{MLM}}(\phi_{t-1})$ is the aggregated posterior $p^{\mathrm{agg}}(\theta)$:

$$\begin{aligned}
p^{\mathrm{agg}}(\theta) &:= \arg \max_{p(\theta|\phi_{t-1})} \mathcal{L}_{t-1}^{\mathrm{MLM}}(\phi_{t-1})\\
&= \arg \max_{p(\theta|\phi_{t-1})} \sum_{i=1}^{t-1} \int_\theta q(\theta|x_i) \log \frac{p(x_i|\theta)p(\theta|\phi_{t-1})}{q(\theta|x_i)}\\
&= \frac{1}{(t-1)} \sum_{i}^{t-1} q(\theta|x_i). && (5.29)
\end{aligned}$$

This result allows us to further rewrite the first and the last term in (5.28), leading to the following greedy objective function at time t that does not depend on previous data:

$$\begin{aligned}\mathcal{L}_t^{\text{Greedy}}(\phi) &= \int_\theta q(\theta|x_t)\log\frac{p(x_t|\theta)p(\theta|\phi)}{q(\theta|x_t)} - (t-1)D_{\text{KL}}[p(\theta|\phi_{t-1})\|p(\theta|\phi)] \\ &= \mathbb{E}_{q(\theta|\phi)}[\log p(x_t|\theta)] - D_{\text{KL}}[q(\theta|x_t)\|p(\theta|\phi)] \\ &\quad - (t-1)D_{\text{KL}}[p(\theta|\phi_{t-1})\|p(\theta|\phi)], \end{aligned} \tag{5.30}$$

where $\phi_{t-1} := \arg\max_\phi \mathcal{L}_{t-1}^{\text{Greedy}}(\phi)$ is the value of ϕ obtained from the previous step. This is a greedy method since the previous approximate posteriors $q(\theta|x_i)$ for $i \leqslant t-1$ were optimized temporally locally at time i, so they can be different from the global optimal solutions $q_t(\theta|x_i) = p(\theta|x_i, \phi_t)$ that maximize $\mathcal{L}_t^{\text{MLM}}(\phi)$. This is also why, if we use this greedy objective, we can only have an approximation instead of an equality in (5.31).

Like VCL, the optimization involved in the greedy objective (5.30) of ML Mixture code takes the form of a KL regularization in the parameter space. However, it overcomes the two above-mentioned limitations of VCL. First, the prior overhead described in Sec. 5.4.1 is resolved because the mixture distribution is now optimized via maximum likelihood instead of chosen in advance. Second, VCL tries to approximate a fixed multimodal distribution $p(\theta|x^{t-1})$ with a simple parametric distribution (a diagonal Gaussian) $q(\theta|x^{t-1})$, whereas the proposed ML Mixture code optimizes the parametric density function $p(\theta|\phi_{t-1})$ to approximate the aggregated posterior:

$$p(\theta|\phi_{t-1}) \approx \frac{1}{(t-1)}\sum_{i}^{t-1} q(\theta|x_i). \tag{5.31}$$

This approximation is more plausible than (5.18) because the right side of (5.18) is complex and fixed, whereas both sides of (5.31) can be adjusted to be close to each other and thus make the approximation closer.

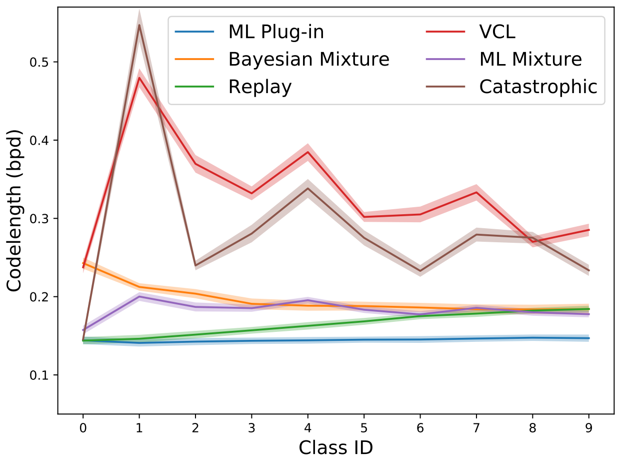
The performance of ML Mixture code in terms of forgetting and compression are also presented in Fig.5.1 and 5.2. One can see that although both VCL and ML Mixture use diagonal Gaussian for the distribution of model parameters (corresponding to $q(\theta|x^{t-1})$ in VCL and $p(\theta|\phi)$ in ML Mixture), ML Mixture achieves much less forgetting and smaller prequential codelength.

5.5.1 *The Intuitive Interpretation*

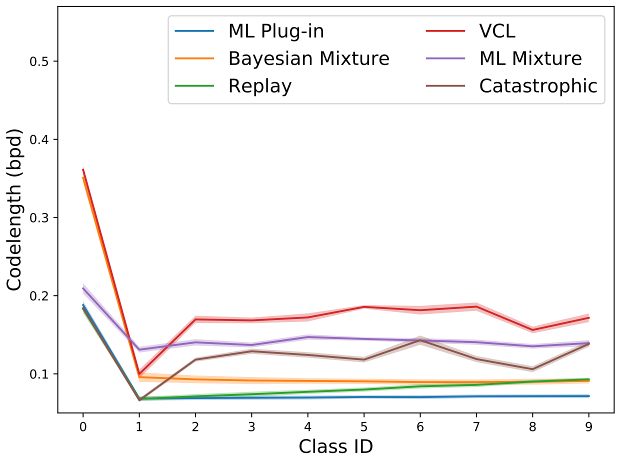
The ML Mixture code is effectively performing ML plug-in coding with a VAE. Unlike normal VAEs, the latent code includes the entire parameter vector of the decoder. This means that the mapping from the latent code to the input space is predetermined by the decoder architecture. Intuitively, ML Mixture projects each data x_i from the input space to a local posterior distribution over the parameter space through the encoder $q(\theta|x_i)$. It then uses the prior distribution $p(\theta|\phi)$ to model the projected data in the parameter space. For each x_i, there exists a large number of simple distributions $q(\theta|x_i)$ that can be used as the encoder. Potentially, one can select the encoder so that the mirrored data points in the parameter space obeys a simple distribution, i.e. $\frac{1}{n}\sum_i^n q(\theta|x_i)$ is a simple distribution. In this case, $p(\theta|\phi)$ which is used to model the mixture distribution can also be kept simple. One supporting evidence for this possibility is the success of normal VAE/WAE [Tolstikhin et al., 2019]. The prior distribution over the joint vector of latent code and the decoder parameters of VAE/WAE is a diagonal Gaussian, if we see the deterministic parameters as Gaussian random variables with zero variance.

5.5.2 *Limitation and Future Improvements*

There are a few limitations of the ML Mixture code that need to be solved before it is practical for continual learning. First, as we greedily optimize only the first and the last terms of (5.28) in a sequential manner, the previous approximate aggregated posterior $p(\theta|\phi_{t-1})$ is fixed once learned. This locks the prior distribution $p(\theta|\phi)$ to a local region biased to the early tasks. In addition, since the first KL term in (5.30) pushes the solution to the new task $q(\theta|x_t)$ close to the prior $p(\theta|\phi)$ biased to solution space of the early tasks, if there are no solutions to the new tasks that are close to the solution space of the early tasks, applying the ML Mixture code will lead to intransigence [Chaudhry et al., 2018], in other words, the model will not be free to learn new tasks well. In Fig. 5.4 - 5.8, we measured the description lengths (in bpd) of all classes encoded by the model trained with different methods throughout the training process (for example, subplot (c) shows the description lengths of images of digit 2 over time), it can be seen that after task 5, with ML Mixture code, the description lengths only improve marginally. A potential fix is to introduce a certain level of generative replay to re-optimize part of the previous loss $\mathcal{L}_{t-1}^{MLM}(\phi_{t-1})$. Second, both VCL and ML Mixture use approximate objectives to remove dependency on previous data. Although (5.31) could be more plausible than (5.18), the error introduced in the objective could still be arbitrarily large as pointed out earlier in Sec. 5.4.2. In the future, it would be preferable to develop an alternative objective for which the approximation error is bounded.

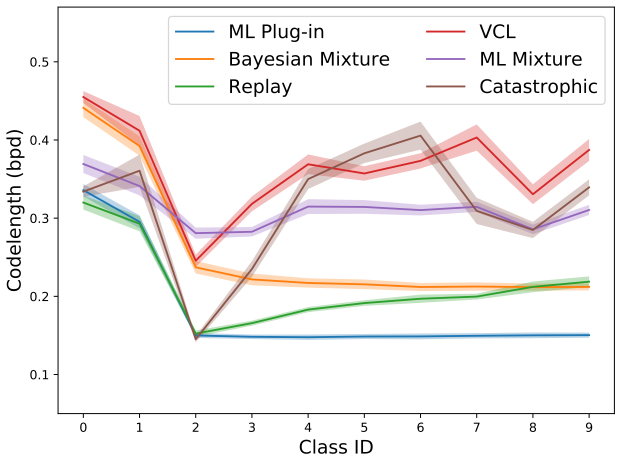


Digit 0

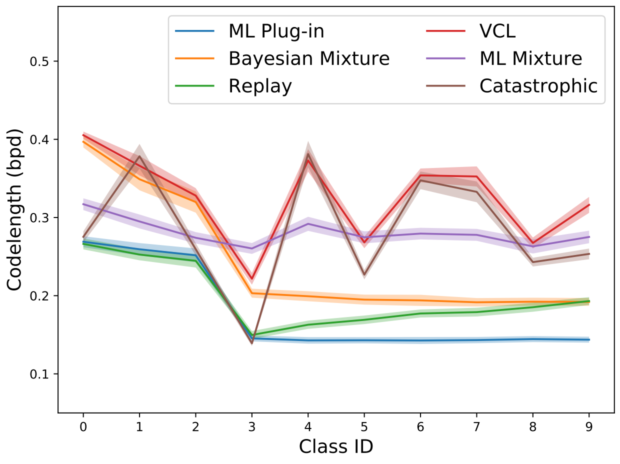


Digit 1

Figure 5.4: The change of description lengths of digits 0 and 1 throughout the training process.

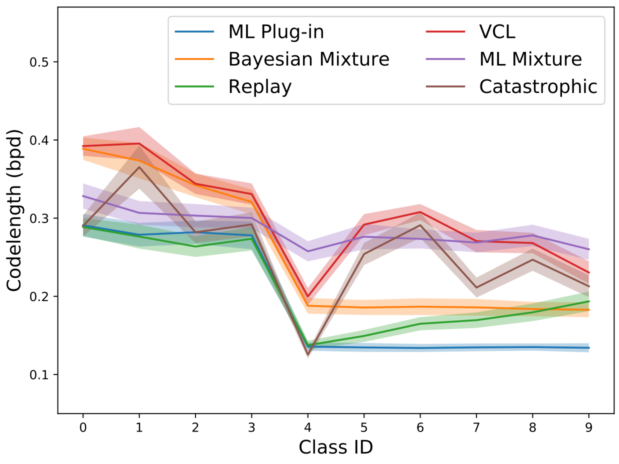


Digit 2

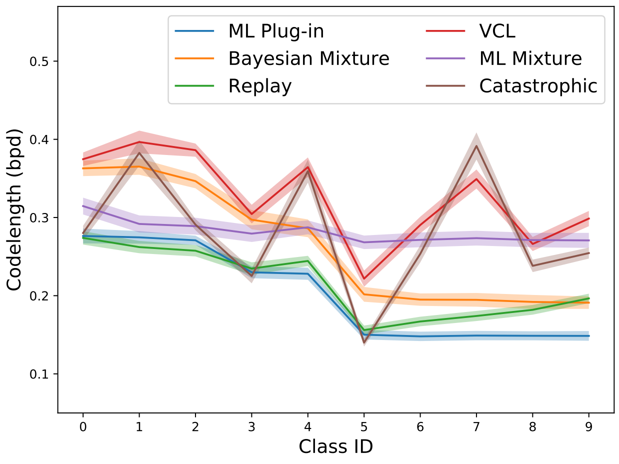


Digit 3

Figure 5.5: The change of description lengths of digits 2 and 3 throughout the training process.

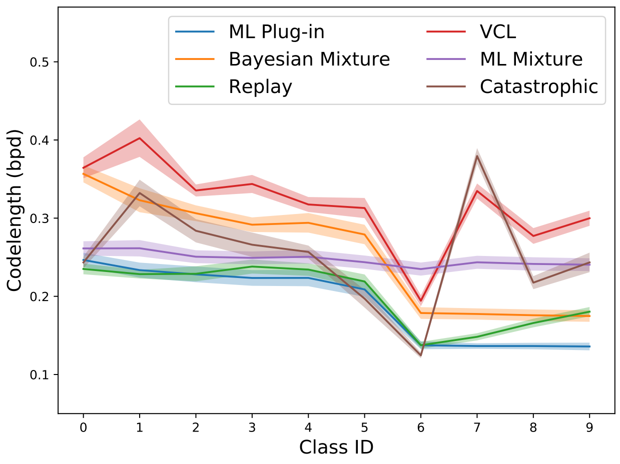


Digit 4

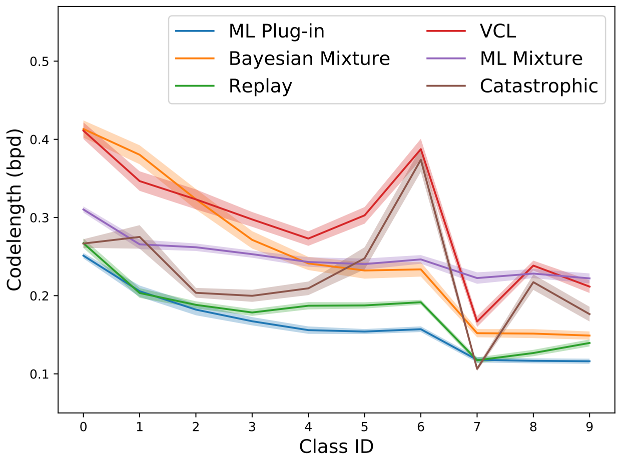


Digit 5

Figure 5.6: The change of description lengths of digits 4 and 5 throughout the training process.

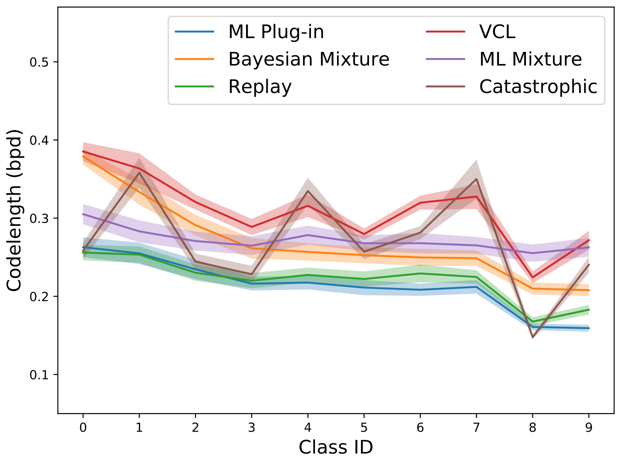


Digit 6

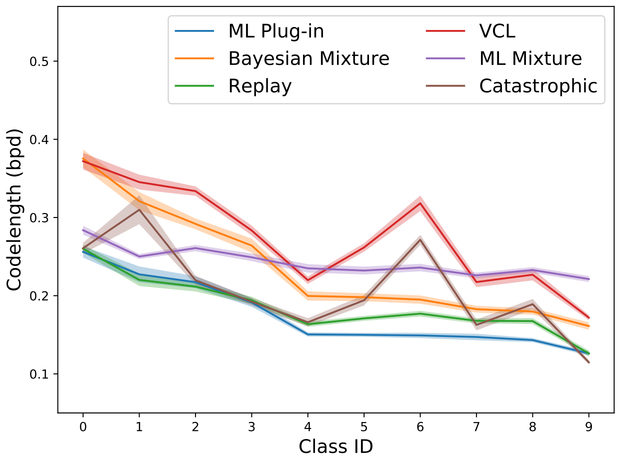


Digit 7

Figure 5.7: The change of description lengths of digits 6 and 7 throughout the training process.



Digit 8



Digit 9

Figure 5.8: The change of description lengths of digits 8 and 9 throughout the training process.

6 RESERVOIR TRANSFER ON ANALOG NEUROMORPHIC HARDWARE

Reservoir computing [Jaeger, 2001; Maass, Natschläger, and Markram, 2002] (RC) is a computational paradigm originated from research on recurrent neural networks that process input signals by transforming them into high-dimensional features via the dynamics of a randomly created, non-linear recurrent network called a reservoir. Instead of adapting the recurrent connections, RC usually keeps them fixed and focuses on only learning a linear readout that maps the reservoir activation to the desired target signals. For this reason, RC has the advantage of being computationally cheap, easy to implement and suitable for online adaptations. Despite of its simplicity, it can often achieve competitive performance on various signal processing tasks. Moreover, it can be applied to a wide range of complex black-box systems by harnessing their intrinsic non-linear dynamics. This renders RC a popular framework for non-conventional computing substrates such as neuromorphic hardware. However, creating a good reservoir in neuromorphic devices can be challenging due to the following constraints imposed by the hardware: device and time scale mismatch, approximate neuron models, unobservable and non-differentiable dynamics and low bit resolution. To overcome these challenges, in this chapter, we take inspirations from the forward transfer idea in continual learning and propose an algorithm called Reservoir Transfer (RT). To achieve forward transfer, CL methods usually apply transfer learning techniques such as knowledge distillation [Hinton, Vinyals, and Dean, 2015] to distill and transfer the knowledge of existing networks to a new network by forcing the student network to map input vectors to the same outputs of the teacher networks. RT shares the same idea and extends it to recurrent networks. With the help of RT, we are able to successfully learn ternary weights in a reservoir of physical spiking neurons by transferring features from a well-tuned echo state network simulated on a digital computer. Empirical results demonstrate that the proposed method is able to train a well-performing recurrent spiking network without precise calibration of individual neurons or using Back-propagation Through Time (BPTT). The Reservoir Transfer method described in this chapter was first proposed in [He, 2018b]. The rest of this chapter is largely a verbatim copy of [He et al., 2019a][1] with an additional review of forward transfer in CL to illustrate its connection to RT.

1 This work received the Best Paper Award, Third Place at the 9th International IEEE Conference on Neural Engineering (NER), 2019.

6.1 INTRODUCTION

Implantable or wearable biosensors, signal generators, controllers or bidirectional brain-machine interfacing modules are widely investigated. For such devices, a very small energy consumption is always desirable and sometimes mandatory – for instance, in brain implants that must not dissipate noticeable heat [Birmingham et al., 2014]. A promising route toward such devices is to develop analog, unclocked, spiking neuromorphic microchips implementing artificial neural networks (ANNs). These ANNs obtain their functionality by being *trained* rather than being *programmed*, using methods from machine learning. With regards to the latter we perceive two main trends. First, adapt *deep learning* methods [Goodfellow, Bengio, and Courville, 2016] to spiking neural networks (for instance [Tsai et al., 2017]). Second, apply computational mechanisms known from biological brains to neuromorphic hardware [Indiveri and Liu, 2015]. Here we follow the second route and employ principles of *reservoir computing* to realize a heartbeat abnormality detector on an analog spiking neuromorphic device.

Reservoir computing (RC) is a learning paradigm in recurrent neural networks (RNNs) for processing temporal input signals. The core RC principles have been discovered several times independently, both in AI/machine learning and in computational/cognitive neuroscience (brief history reviewed in [Jaeger, 2007]), indicative of the nature of RC as bridging between machine learning and neuroscience. Within computational neuroscience, RC is best known as *liquid state machines* (LSMs) [Maass, Natschläger, and Markram, 2002], whereas the approach is known as *echo state networks* (ESNs) [Jaeger and Haas, 2004] in machine learning. Reflecting the different objectives in these fields, LSM models are typically built around more or less detailed, spiking neuron models with biologically plausible parametrizations, while ESNs mostly use highly abstracted rate models for its neurons.

An RC architecture is composed of three major parts: the *input layer* feeds the input signal into a random, large, fixed recurrent neural network that constitutes the *reservoir*, from which the neurons in the *output layer* read out a desired output signal. In traditional (and "deep") RNN training methods, all synaptic weights in a neural learning architecture are optimized by descent on the output error gradient. In contrast, the input-to-reservoir and the recurrent reservoir-to-reservoir weights in an RC system are left unchanged after a random initialization, and only the reservoir-to-output weights are optimized during training – typically by minimizing the mean square error between the teacher signal and the network's output signal through linear regression.

In the work reported here, we sought to create a functional spiking reservoir on an analog, unclocked, spiking neuromorphic microprocessor board. The system's task is to detect and classify heartbeat abnormalities from electrocardiographic (ECG) input signals. The system should function in an online processing mode in real-time. Our hardware is the Dynap-se board [Moradi et al., 2017]. This work constituted a work package in the European Horizon 2020 project NeuRAM3 (`http://neuram3.vsos.ethz.ch/`), which is concerned with the design and fabrication of memristor-based, spiking neuromorphic microchips.

6.2 DYNAPSE BOARD

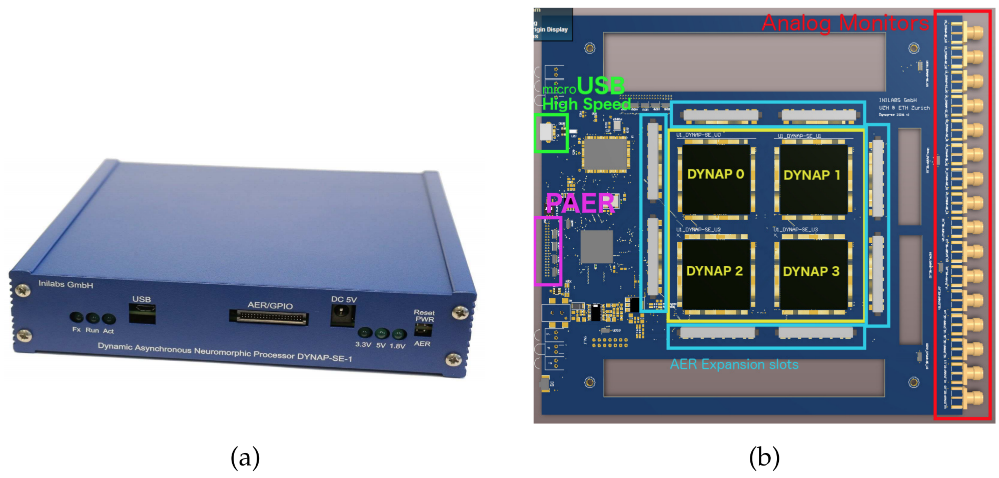


Figure 6.1: The DYNAP-se board used for this project. (a) The exterior appearance of the board, it comes with a USB interface to standard personal computers. (b) The internal layout of the board. It consists of four DYNAP chips: DYNAP 0-3. Images were taken from Dynap-se user guide [*Dynap-se user guide* 2017].

The working device of our project is a unclocked analog spiking microchip called DYNAP-se (Fig. 6.1 (a)) [Moradi et al., 2017], which stands for Dynamic Neuromorphic Asynchronous Processor (Scalable). The Dynap-se board we used contains four chips (see Fig.6.1), each of the four neuro-chips contains four cores with 256 neurons each (see Fig. 6.2). These on-chip neurons are designed based on the Adaptive Exponential Integrate-and-Fire (AdEx) model [Brette and Gerstner, 2005], which generalizes the leaky integrate-and-fire (LIF) model [Burkitt, 2006]. The behavior of one neuron is characterized by 25 parameters such as injection current level, refractory period length, time constants, and synaptic efficacy. These parameters can be set globally for each of the cores but not on the individual neuron level. Due to device mismatch, effective values of these parameters vary across a neuron population. As a result, although we can specify the same parameter bias for all neurons on Dynap-se, every individual

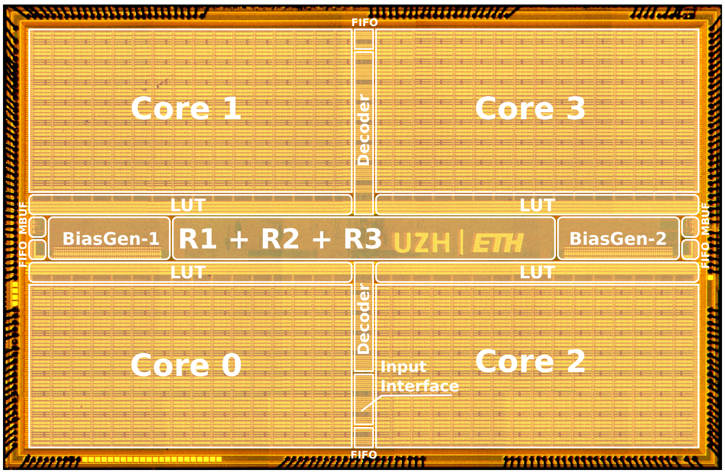


Figure 6.2: A photo of the DYNAP multi-core neuromorphic processor. The chip consists of four cores, with 256 neurons each. Neurons in different cores or different chips can send spikes to each other via the routers: R1-R3. The dynamics of the neurons and synapse can be specified by the bias generators (BiasGen-1 and BiasGen-2) built on-chip. Image source: [Moradi et al., 2017]

neuron exhibits very different behavior, as reported in the Dynap-se user guide [Qiao et al., 2015]. Moreover, the connections between neurons are restricted to only ternary values $+a, -a, 0$ (the scaling factor a can be set per core), corresponding to excitatory, inhibitory or no connections. The synapic connections are stored as content-addressable memory (CAM) on the chip. Each neuron can have up to 64 CAMs for pre-synaptic connections and unlimited post-synaptic connections. Finally, since the chip is an analog device, state variables such as membrane potentials and post-synaptic currents are only observable through an oscilloscope, which can be connected to the analog monitor outlet marked in Fig. 6.1 (b). However, these state variables cannot be recorded. Therefore, for information processing and learning, only spike trains can be recorded and used. These spike trains can be recorded by a standard PC connected via the USB interface shown in Fig. 6.1 (a).

6.3 THE CHALLENGES

Our project encountered interesting difficulties that arose from its dual nature of being half engineering/machine learning, half bio-/neuroscience:

Machine learning: We want to achieve numerically accurate and robust information processing performance, goals of which are characteristic for machine learning rather than for computational neuroscience. This would suggest using the machine learning version of RC, namely ESNs. Indeed we found that the addressed processing task could be solved by the standard ESN methods with high performance. However, standard ESN methods rely on floating-point precision and reproducible arithmetics, and ESN neuron models are non-spiking and thus entirely inappropriate to capture Dynap-se neurons.

Bio-/neuroscience: The physical neurons on the Dynap-se board share core properties with biological neurons. They are spiking, unclocked, individually different from each other ("device mismatch"), (slightly) noisy, and must be (approximately) modeled by rather intricate, multi-parameter ODEs with several time constants. Furthermore, the input ECG signal is of biological origin and must be processed with the appropriate biological timescales. All of this would indicate to use LSM models. The LSM literature offers numerous successful accounts of modeling biological neural systems. However, these reports did not provide useful guidance for us because first, they rest on approximate copies of biological systems whose neuron models and parameters (especially time constants) are incommensurate with our physical device, and second, LSM literature reports are typically proof-of-concept studies whose numerical accuracy or robustness would not match our needs.

Additional challenges result from the circumstance that many physical variables (currents and potentials) determining the behavior of Dynap-se neurons are not measurable – a condition familiar to neuroscientists but alien to machine learning programmers.

In summary, we were suspended between two well-proven computational paradigms of which we would want to use the first for accuracy and robustness, the other for matching our hardware, but none of the two could easily be accommodated.

The solution came from an unexpected side. In our initial investigations, we found ourselves confronted with yet another problem which we did not anticipate: a mismatch of timescales. Our Dynap-se hardware was far too fast for processing our slow ECG signals in a time-coupled online fashion. Classifying a heartbeat as normal vs. abnormal requires to integrate information over a timespan in the order of 1 second. Even when the few programmable time constants on the Dynap-se board were set to the slowest possible values, the Dynap-se dynamics was still inherently too fast to realize the requisite real-time memory spans.

At this point, we recall some facts about short-term memory in RNNs. There are two complementary mechanisms by which information is preserved through time in an RNN (or, in fact, in any input-driven dynamical system modeled by ODEs):

- The model's timescale can be set by the time constants of the governing equations. With large (slow) time constants, the effects of an input signal $\mathbf{u}$ will only slowly "wash out" over time. However, large time constants amount to smoothing the effects of the input signal on the current system state. In our concrete case, if we would induce slowness solely by slow time constants, any characteristic of the input signal $\mathbf{u}(t_0)$ which is relevant for the classification at time $t_0 + 1$ (sec) would become overshadowed by almost equally preserved but distracting information from earlier input times.

- The second mechanism is inherent in the nonlinear geometry of dynamical systems. In its core, it is captured by extensions of Takens' theorem [Stark, 1999; Stark et al., 2003]. Roughly speaking, in any dynamical system (continuous or discrete time) some portion of the preceding input and internal state history is nonlinearly encoded in the current system state, such that parts of the previous input history can be recovered from the current state. This effect has been investigated in depth in the RNN literature in general and the ESN literature in particular (for instance [Ganguli, Huh, and Sompolinsky, 2008; Hermans and Schrauwen, 2010]). This dynamical short-term memory capacity is unfortunately quite sensitive to system noise.

To make the most of the first mechanism, we discovered a few heuristic tricks to set the biases of the on-chip neurons and synapses in order to alleviate the timescale mismatch problem. For example, we used slow synapses for recurrent connections and set the parameters for the time constants of neurons to their slowest possible values (a detailed account of these heuristic parameter settings and the reasons behind them can be found in [Liu, 2019]). These heuristic solutions were able to slow down the dynamics of the on-chip neural network. Their effects are evidently demonstrated in Fig. 6.3, where we created two reservoirs (baseline and tuned) with the same network topology but different parameters for the bias generators. The baseline reservoir are configured with the default bias given by the software interface for Dynap-se called NetParser[2]. The tuned reservoir, on the other hand, used the parameters obtained by the heuristics we discovered. To both reservoirs, after an initial silence of 0.5 seconds, we sent a pulse of spikes for 1 second, then we stop sending any input for 5 seconds. Every two consecutive spikes in the 1-second spike train are separated by only 0.001 seconds. The responses (output spikes) of the on-chip neurons were recorded and smoothed by an exponential-decay kernel, Fig. 6.3 visualizes the responses of 100 randomly picked neurons from each reservoir. It can be seen clearly that the neuron activity in the baseline reservoir started to decay to silence as soon as we stopped sending input spikes, whereas the tuned reservoir had long-lasting activation and the firing patterns were also more diverse.

2 https://gitlab.com/aiCTX/caer-modules-dynapse/

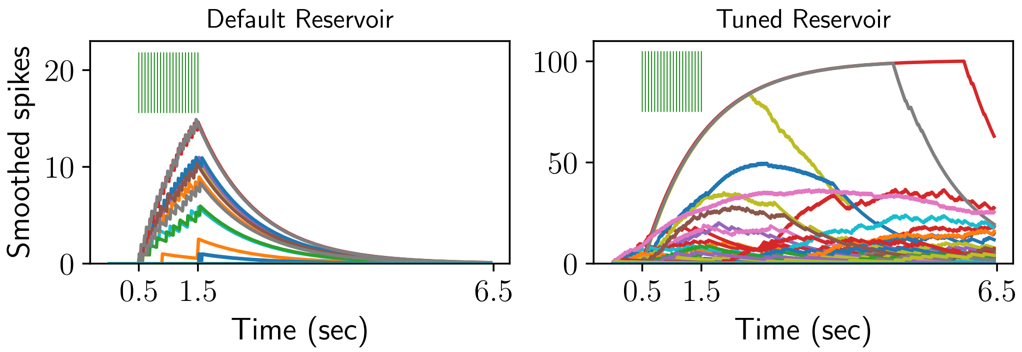


Figure 6.3: The exponentially smoothed reservoir responses to a one-second pulse of spike trains. 100 neurons were randomly selected from each of the reservoirs. The input spike trains are visualized with green vertical bars. Left: responses of the default reservoir. Right: responses of the tuned reservoir. Image source: [Liu, 2019].

After tuning the bias generators to slow down the individual synapses and neurons, we capitalized on the second mechanism to bridge the remaining memory gap into the required 1 second time range. To this end we first optimized the Takens-type dynamic memory characteristics of a standard ESN with leaky integrator neurons using a digital computer, employing routine methods from the ESN field. The optimized dynamics of this reservoir was then "mirrored" in a hardware reservoir on the Dynap-se board with the same number of neurons, in a novel, local, neuron-by-neuron training scheme that relied on linear regression. In this hardware training process, the reservoir-to-reservoir synaptic weights on the Dynap-se board were determined in a way that made the hardware reservoir inherits the (slow, input-history preserving) dynamical characteristics of the optimized source ESN. By using a regularized version of linear regression, the noise robustness of the hardware reservoir was optimized as a side-effect.

In summary, solving the slow timescale problem resulted in a *Reservoir Transfer Method* which effectively built a bridge from the engineering/machine learning world of ESNs to the bio/neuroscience world of LSM-like systems. Before we present the technical details in Section 6.5, we outline the connections of our approach with forward transfer in continual learning.

6.4 FORWARD TRANSFER IN CONTINUAL LEARNING

Forward transfer refers to the phenomena that learning a similar task in the past has positive influence on the performance of tasks in the future. In order to achieve forward transfer, a continual learning agent should be capable of preserving the knowledge from the past. This can be done by distilling the knowledge of an older model from the past and transferring it to the current model. This problem of transferring knowledge from one model to another without much loss of validity is thoroughly studied in the field of model compression, which was introduced by Buciluǎ, Caruana, and Niculescu-Mizil [2006]. They observed that even though ensembles of a large amount of base-level classifiers can achieve good performance, they often require a large space to store the models and a lot of time to execute the computation during inference. On the other hand, unlike the unknown true function, the high performing ensemble model is known and can be used to generate large amounts of pseudo-data. They also showed that with enough pseudo-data as supervision, a smaller and faster model can approximate the large complex ensemble well with little loss of performance. This idea was later generalized and popularised by the Knowledge Distillation technique proposed by Hinton, Vinyals, and Dean [2015]. Contrary to the traditional view of machine learning, a large over-parameterised deep neural network tends to generalize better than a smaller network when they are both trained on the same dataset with the same computational resources [Gou et al., 2021; Neyshabur et al., 2018]. This motivates deep learners to use increasingly larger models for greater performance [Brown et al., 2020; Devlin et al., 2019]. However, this also renders most of these high-performing models too cumbersome to be deployed for many use cases such as real-time or on-device applications, which have stringent requirements on latency and computational resources. On the other hand, even though directly training smaller networks on the same dataset often leads to poor generalization, they actually have enough capacity to approximate the same functions as the larger models. For this reason, knowledge distillation was designed to distill the knowledge from a large high-performing teacher network to a smaller faster student network. The main idea of knowledge distillation is to use the softmax class probabilities produced by the teacher network as the soft targets for the student network as these probabilities contain much more information than the hard targets. Via distillation, the smaller network is trained to generalize in the same way as the larger network, hence it will typically perform much better on test data than trained on the original data directly.

In Continual Learning, this idea of transferring knowledge by matching the outputs with a teacher model is also widely adopted. For instance, in Deep Generative Replay [Shin et al., 2017], the authors trained a deep generative model ("generator") in tandem with a task solving model ("solver") on a sequence of tasks. For every new task, a new solver is created and trained to replace the previous solver and it preserves the knowledge of the old solver by matching the outputs on the same sampled inputs from the generator during the replay process. Expansion-based continual learning approaches grow the size of the model as more tasks are learned, thus some of these approaches apply this transfer technique to compress the model in order to maintain a bounded size. Schwarz et al. [2018] proposed a CL framework called Progress and Compress, which consists of two components: a knowledge base, capable of solving all previous tasks, and an active column network that has lateral connections from the knowledge base and can efficiently learn the current task. During the progress phase, the new task is quickly learnt by the active column reusing features from the knowledge base. During the compress phase, the new knowledge of the active column is distilled into the knowledge base by minimizing the KL divergence between the output probabilities of the active column and those of the knowledge base.

In the next section, we propose a knowledge transfer technique that extends this idea of function matching to dynamics matching in recurrent neural networks and apply it to creating a functional reservoir in a spiking neuromorphic device.

6.5 RESERVOIR TRANSFER METHOD

THE TEACHER RESERVOIR We use an ESN with leaky integrator neurons as the teacher reservoir for this work. This ESN is driven by some m-dimensional input signal $\mathbf{u}(t)$ and its dynamics are described as follows

$$\dot{\mathbf{x}}(t) = -\lambda_x \mathbf{x}(t) + \tanh(W\mathbf{x}(t) + W^{in}\mathbf{u}(t)), \tag{6.1}$$

where $\mathbf{x}(t)$ is the n-dimensional state vector of the network at time t, λ_x the leaking rate, $W^{in} \in \mathbb{R}^{n\times m}$ and $W \in \mathbb{R}^{n\times n}$ are input and recurrent weights, respectively. In RC, these weight matrices are first randomly initialized then scaled according to some global parameters such as the dimension of W^{in} or the spectral radius of W [Lukoševičius, 2012]. Since this teacher reservoir is simulated in a digital computer, we can use high precision weights and all its state variables are available for training and evaluation. So we can guarantee it is a well-performing reservoir before we use it for the reservoir transfer process. The state trajectory $\mathbf{x}(t)$ can be considered as the high-dimensional temporal features of $\mathbf{u}(t)$, which we would like to transfer to a student reservoir.

THE STUDENT RESERVOIR The spiking network that we use as the student reservoir in this work is a recurrent network of leaky integrate-and-fire (LIF) neurons, whose dynamics are approximately modeled by:

$$\dot{\mathbf{v}}(t) = -\lambda_v \mathbf{v}(t) - \theta\mathbf{s}(t) + \hat{W}\mathbf{r}(t) + \hat{W}^{in}\mathbf{u}(t) + I, \tag{6.2}$$

$$\dot{\mathbf{r}}(t) = -\lambda_s \mathbf{r}(t) + \mathbf{s}(t), \tag{6.3}$$

where $\mathbf{v}, \mathbf{s}, \mathbf{r}$ are all n-dimensional vectors whose i-th entries are denoted as v_i, s_i and r_i, respectively. v_i is the membrane potential of the i-th neuron, and $s_i(t) = \sum_{t_s} \delta(t - t_s)$ is its output spike train with spike times t_s and Dirac delta function $\delta(\cdot)$. A LIF neuron spikes when its membrane potential v_i reaches a firing threshold θ^f, then v_i

is immediately reset to a resting potential θ^r. θ in (6.2) is the difference between spiking threshold and reset potential: $\theta = \theta^f - \theta^r$, r_i is the exponentially decaying synaptic currents triggered by s_i. I is a constant current set near or at the rheobase (threshold to spiking) value, as used in [Nicola and Clopath, 2017]. $\hat{W}^{in} \in \mathbb{R}^{n \times m}$ is the input weight matrix of the spiking reservoir and $\hat{W} \in \mathbb{R}^{n \times n}$ is the recurrent weight matrix, whose value will be learned in the following transfer process.

MIRROR THE DYNAMICS To transfer the temporal features $\mathbf{x}(t)$, we inject the ESN feature signals $x(t)$ element-wisely into the corresponding LIF neurons, replacing the recurrent inputs $\hat{W}r(t)$. This results in the dynamics:

$$\dot{\mathbf{v}}_x(t) = -\lambda_v \mathbf{v}_x(t) - \theta \mathbf{s}_x(t) + \mathbf{x}(t) + W^{in}\mathbf{u}(t) + I, \tag{6.4}$$

$$\dot{\mathbf{r}}_x(t) = -\lambda_s \mathbf{r}_x(t) + \mathbf{s}_x(t). \tag{6.5}$$

Ideally, we would like the same dynamics $\mathbf{v}_x(t)$ to be sustained by the recurrent input $\hat{W}\mathbf{r}(t)$ instead of the teacher signal $\mathbf{x}(t)$. In other words, if we consider $\mathbf{y}(t) := \hat{W}\mathbf{r}_x(t)$ as the state vector of the spiking reservoir, we aim to learn the weight matrix $\hat{W}$ such that $\mathbf{y}(t) = \mathbf{x}(t)$. In order to make the learned weights generic and input-independent, we choose $\mathbf{u}(t)$ to be a white noise signal for training, and compute the weights by linear regression to minimize

$$\mathcal{L}^{\text{Transfer}}(\hat{W}) := \sum_{t_k} ||\hat{W}\mathbf{r}_x(t_k) - \mathbf{x}(t_k)||_2^2, \tag{6.6}$$

where t_k are discrete time samples. In this way, we have turned the weight initialization problem into a linear regression problem.

This reservoir transfer method can be easily generalized to other type of recurrent networks. For instance, the teacher reservoir can be a trained Long Short-Term Memory (LSTM) network [Hochreiter and Schmidhuber, 1997] or another spiking network with different neuron models. The student network can also be an artificial recurrent network. In addition, for simplicity, we used the same number of neurons for teacher and student reservoirs in the descriptions above. However, this method is not limited to only such cases. One neuron in the teacher reservoir can correspond to multiple student neurons if the student reservoir has more neurons. In the case that the teacher network has more neurons, a subset of representative neurons from the teacher network can be selected to teach the student reservoir.

Figure 6.4 shows the dynamics $\mathbf{y}(t) = \hat{W}\mathbf{r}(t)$ of a reservoir of 100 LIF neurons created in the Brian simulator [Stimberg, Brette, and Goodman, 2019] using this method and the dynamics $\mathbf{x}(t)$ of its teacher leaky ESN reservoir of the same size when they are driven by a sinusoidal input. Only the first four dimensions of $\mathbf{x}(t)$ and $\mathbf{y}(t)$ are displayed

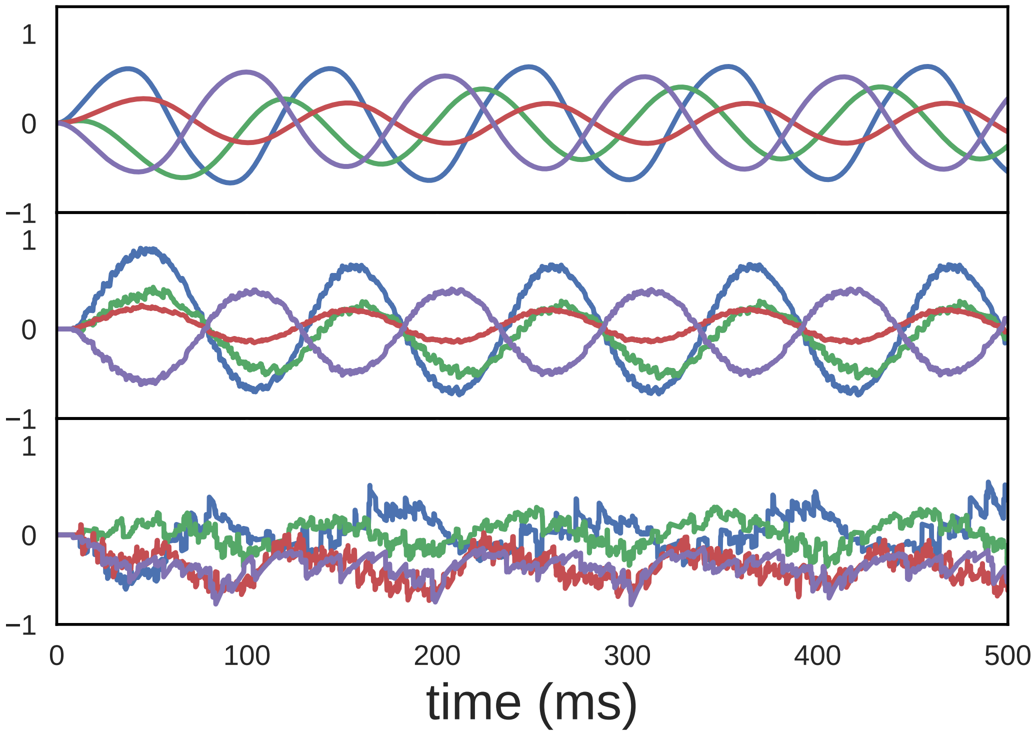


Figure 6.4: Dynamics of different reservoirs in response to a sinusoidal input. For visual simplicity, only the first four dimensions of the state vectors are displayed. **Top**: $\mathbf{x}(t)$ in a leaky ESN teacher reservoir. **Middle**: corresponding $\mathbf{y}(t)$ in the target LIF reservoir created using the reservoir transfer method. **Bottom**: corresponding $\tilde{\mathbf{y}}(t)$ in a reservoir whose recurrent weights are from a sparse matrix with randomly distributed values (for explanation, see text).

for visual simplicity. One can see that although the reservoir was trained using white noise input, its similarity to the teacher reservoir can generalize to other types of input signals (a sinusoidal wave in this case). For comparison, we also created a LIF reservoir whose recurrent weights $\tilde{W}$ are from a sparse matrix with values first randomly sampled from a standard normal distribution, then scaled by a constant so that the recurrent dynamics do not evoke constant bursts of spikes. The first four dimensions of its response $\tilde{\mathbf{y}}(t) = \tilde{W}\mathbf{r}(t)$ to the same sinusoidal input is shown in the bottom panel of Figure 6.4.

6.6 EXPERIMENTS

We empirically evaluate the transferred reservoirs by experiments with both software simulation and hardware experiments.

6.6.1 *Short-term Memory of Transferred Reservoir*

In order to validate that the reservoir generated from the transfer learning method has short-term memory despite the connections are of low bit-precision and there is no synaptic short-term plasticity, we created such a reservoir in the Brian simulator [Stimberg, Brette, and Goodman, 2019]. Instead of using the standard ridge regression during transfer learning, we used ternarized linear regression [Zhu et al., 2017] so that the resulting weights are of ternary precision: $-a, 0, a$. To test its short-term memory, a sequence of pulses with very short pulse widths (10 ms) separated by long (200 ms) periods of silence was used as input to drive the ternary reservoir. A linear full precision readout was then trained to map the filtered spikes to a reverse-chirp signal. The time constant of the exponential decay kernel used to filter spike trains is only 15 ms. Since the output depends on the past input values, it can only be possible if the reservoir preserves some information about the input history. Figure 6.5 shows the step signal, the output of the reservoir and its target during the testing phase.

6.6.2 *ECG Monitoring using Dynap-se*

In this experiment, we exploited our transferred spiking reservoir in a patient-customized electrocardiogram (ECG) heart beat classification task realized on Dynap-se. To circumvent the constraints imposed by the hardware, we implemented the above-mentioned Reservoir Transfer method to create a reservoir using 3 cores on Dynap-se. A leaky ESN of equal size was created in simulation as a teacher reservoir, and its response to white noise input signal $\mathbf{u}(t)$ was collected and converted into spike trains to be sent to the target reservoir in Dynap-se. After the output spike trains from the hardware

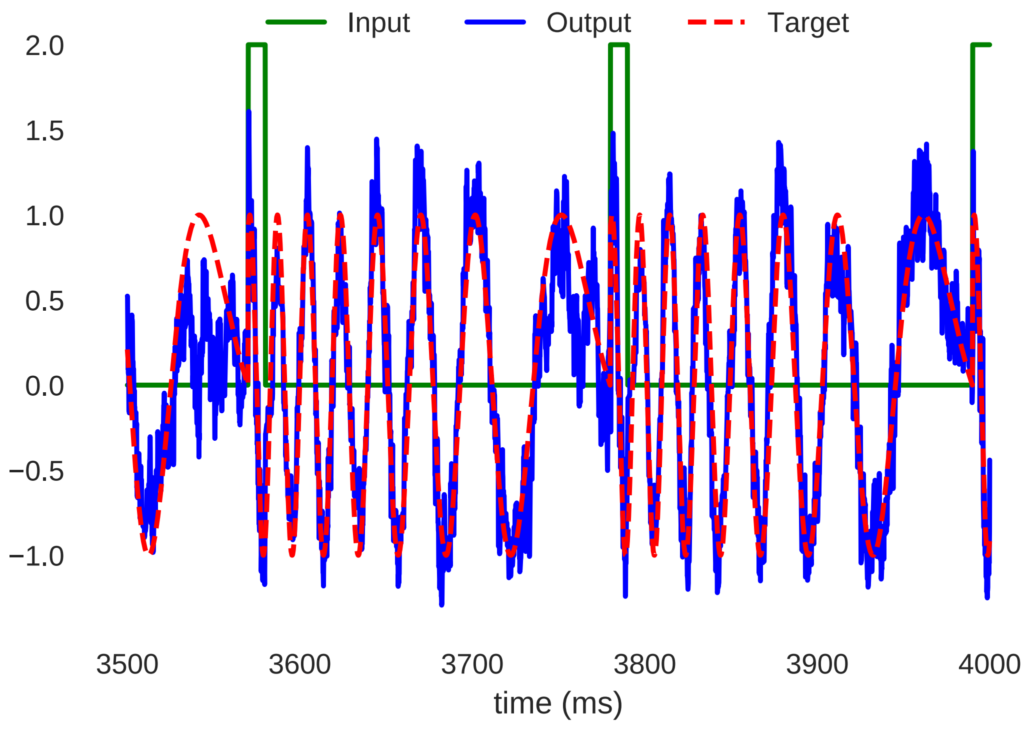


Figure 6.5: A reverse-chirp signal can be generated by a reservoir of LIF neurons with ternary weights when it is driven by a step signal with very short (10 ms) high signal followed by a long (200 ms) silence.

neurons are recorded, we smooth both the input and output spike trains by an exponential decay kernel to get $\mathbf{x}(t)$ and $\mathbf{r}(t)$, respectively. Finally, ternarized linear regression [Zhu et al., 2017] was applied to compute the ternary weight matrix W_{ternary} such that $\mathbf{x}(t) \approx W_{\text{ternary}}\mathbf{r}(t)$.

In this way, we can circumvent the problem that the internal variables of individual neurons cannot be recorded and calibrated: there is no need to know the exact values of the membrane potentials and other parameters, because all we need to run this algorithm are the input and output spike trains, which are accessible from the Dynap-se board. The neurons do not have to share the same parameter value as long as their collective response to the input current $\mathbf{x}(t)$ contains enough information to linearly decode $\mathbf{x}(t)$. The linear regression algorithm will automatically adapt to their variety, hence it also addresses the problem of device mismatch. Moreover, learning is needed only once using a white noise signal, afterwards the connection weights can stay fixed. Hence no online adaptation on hardware is needed.

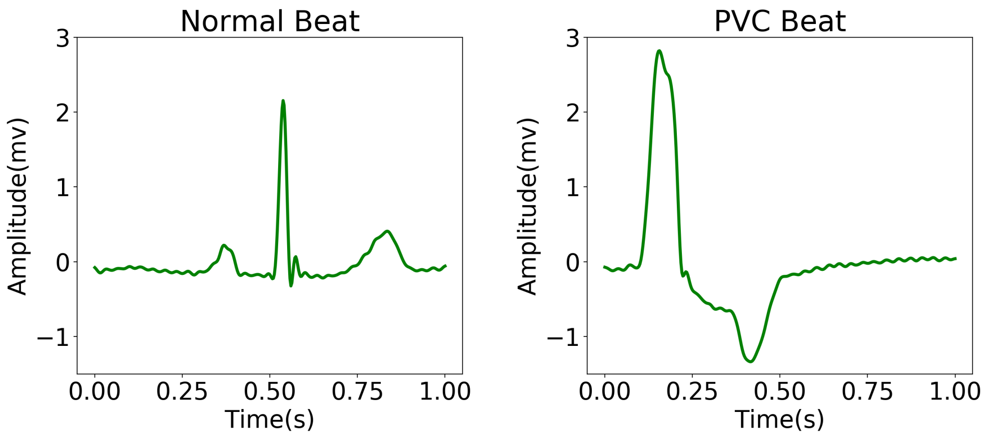


Figure 6.6: Left panel: a normal heartbeat. Right panel: a PVC heart beat.

To verify that the transfer learning method yields a functional physical spiking reservoir, we conducted an ECG signal classification experiment using the learned reservoir on Dynap-se. The experiment aims to detect Premature Ventricular Contractions (PVCs), which are abnormal heartbeats initiated by the heart ventricles. Figure 6.6 shows a normal heartbeat and a PVC. More concretely, we formulate the PVC detection task as a supervised temporal classification problem which demands a binary output signal $z(n)$ for each heartbeat indexed by n:

$$z(n) = \begin{cases} 1 & \text{if the } n\text{-th heartbeat is a PVC,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.7}$$

The experiment was conducted with the following routine.

1. *ECG pre-processing*: we removed the baseline drift from an ECG signal by applying a high-pass Butterworth filter and then normalized the signal into the numerical range [0,1].

2. *Signal-to-spike conversion*: we placed a spike at a time index if the increase/decrease of the ECG signal relative to its value at the previous spike time surpassed a threshold of numerical value 0.1.

3. *Reservoir response harvesting*: we sent ECG-converted spike trains into Dynap-se to harvest the reservoir responses, which were also in the form of spike trains.

4. *Spike-to-signal conversion*: on a digital computer, we smoothed the spike trains collected from the physical reservoir to continuous-valued time-series by an exponential decay kernel with a decay time constant 3.5.

5. *Classifier training*: the training of the readout mechanism amounted to solving a linear regression problem, where the input for linear regression was the smoothed reservoir responses and the target output was a $\{0, 1\}$-valued binary signal indicating the correct labels of heartbeats.

6. *Result evaluation*: with a testing ECG time-series, we repeated the above procedure to procure its smoothed reservoir responses and then readout the predicted labels with learned weights. The predictions are then compared to ground truth labels to evaluate the classification performance. The metrics we used are described in details below.

We used the following standard metrics for evaluating binary classification tasks: accuracy (Ac), sensitivity (Se), precision (P), and F1-score:

$$\begin{aligned}
\mathrm{Ac} &= \frac{(\mathrm{TP}+\mathrm{TN})}{(\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN})},\\
\mathrm{Se} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}},\\
\mathrm{P} &= \frac{\mathrm{TP}}{(\mathrm{TP}+\mathrm{FP})},\\
\mathrm{F1} &= \frac{(2\mathrm{TP})}{(2\mathrm{TP}+\mathrm{FP}+\mathrm{FN})}.
\end{aligned}$$

where the symbols above denote the following quantities

- TP: the number of true positive predictions (PVC heartbeat detected as PVC).
- TN: the number of true negative predictions (normal heartbeat detected as normal).
- FP: the number of false positive predictions (normal heartbeat detected as PVC).
- FN: the number of false negative predictions (PVC heartbeat detected as normal).

We used MIT-BIH ECG arrhythmia database [Goldberger et al., 2000; Moody and Mark, 2001] in this experiment. The database provides 48 half-hour excerpts of two-channel ambulatory ECG recording files, obtained from 47 different patients. The recordings were digitized with a sampling frequency of 360 Hz and acquired with 11-bit resolution over a 10mV range. Each record was annotated by two or more cardiologists independently, both in timing information and beat classification. In this work, we present the results of a case study where recordings from lead II of file #106, #119, #200, #201, #203, #223, and #233[3]. Since each individual patient has specific features that are different from other patients, it has become a trend in the field of smart healthcare to deliver personalized medical solutions [Khan, Zomaya, and Abbas, 2017]. Therefore, we also customize the classifier for each subject and train a different readout layer for each subject using the corresponding annotation file. The reservoir itself, in other words, the recurrent weight matrix is shared across different subjects. In particular, for each subject, we employed the first 10 minutes of the recording signal for training and the next 5 minutes for testing.

A comparison of classification accuracy on testing data between the low-precision spiking reservoir and the digitally simulated, high-precision reservoir baseline is provided in Table 6.1. Simulation on baseline standard ESN was performed with parameters set as leakage rate = 0.99, spectral radius= 0.9 and regression parameter = 1e-6.

3 We only used the file #119 in [He et al., 2019a], this experiment was later extended to the other subjects in [Liu, 2019].

Table 6.1: PVC detection results on testing data

		Performance Metrics			
subject number	classifier	Accuracy	Sensitivity	Precision	F1
subject #106	Standard ESN	98.75 %	97.22 %	97.22 %	97.22 %
	Dynap-se reservoir	91.30 %	88.89 %	76.19 %	82.05 %
subject #119	Standard ESN	99.70 %	100 %	99.10 %	99.55 %
	Dynap-se reservoir	97.87 %	100 %	94.07 %	96.94 %
subject #200	Standard ESN	99.07 %	98.24 %	99.40 %	98.82 %
	Dynap-se reservoir	95.80 %	93.53 %	95.78 %	94.64 %
subject #201	Standard ESN	99.24 %	100 %	97.18 %	98.57 %
	Dynap-se reservoir	97.74 %	95.71 %	95.71 %	95.71 %
subject #203	Standard ESN	98.14 %	100 %	90.32 %	94.92 %
	Dynap-se reservoir	89.28 %	79.38 %	70.64 %	74.76 %
subject #223	Standard ESN	99.07 %	99.05 %	98.11 %	98.58 %
	Dynap-se reservoir	90.53 %	76.15 %	84.69 %	80.19%
subject #233	Standard ESN	99.78 %	100 %	99.21 %	99.60 %
	Dynap-se reservoir	97.46 %	93.01 %	97.79 %	95.34 %

It can be seen from Table 6.1 that although the Dynap-se reservoir created by the proposed reservoir transfer method performed in general worse than the standard ESN simulated on full bit-precision digital computer, it has achieved reasonably good performance despite its low bit-precision, device mismatch and analog signal noise.

6.7 CONCLUSION

Implementing efficient algorithms on neuromorphic hardware with low energy consumption is a promising yet challenging path towards novel brain-machine interfacing neuro-technologies. In this chapter, we reviewed the major difficulties we encountered along this path: low bit resolution, device mismatch, uncharacterized neural models, unavailable state variables, physical system noise and most importantly, timescale mismatch. As a solution, we proposed a computational scheme called Reservoir Transfer to circumvent these difficulties and created a functional spiking reservoir on the Dynap-se analog asynchronous neuromorphic microprocessor board. Empirical results from an ECG signal monitoring task showed that this reservoir with ternary weights was able to not only integrate information over a time span longer than the timescale of individual neurons but also functioned as an information processing medium with performance close to a standard, high precision, deterministic, non-spiking ESN.

For future work, it remains to be understood how a transferred weight matrix is related to the time constant and leakage rate of the teacher reservoir. In addition, applying this method to transferring trained recurrent neural networks instead of a randomly created ESN will also be an exciting direction for further research.

7 EPILOGUE

Continual lifelong learning is an important aspect of artificial general intelligence. In this thesis, we have approached this goal from multiple angles, and proposed several solutions according to different scenarios of continual learning. In this chapter, we summarize the main contributions of this thesis, point out the limitations of the proposed methods, discuss new understandings and perspectives that came to light towards the end of this Ph.D. program, and finally suggest new directions for future research.

7.1 SUMMARY

We started with the conventional task-aware setting of continual learning, in which a sequence of tasks are presented with clearly defined boundaries during training time. To address the problem of catastrophic forgetting in this setting, we applied the theory of conceptors to define and identify memory space in each layer of a deep network. In particular, we showed that training a neural network on one task does not necessarily exhaust all of its capacity. The remaining power can be capitalized by conceptors. To this end, we have proposed two continual learning methods based on conceptors.

In Chapter 2, we introduced Conceptor-Aided Backprop (CAB), which prevents the weights of a network from forgetting by only exploiting the free space in the input layer during the back-propagation process. The gradients are projected by the conceptors corresponding to the free space, thus leaving the linear mapping from the used space intact. We have also shown that the singular value spectra and the quota of conceptors can be used to monitor how much memory space has been used in each layer of a deep network.

In Chapter 3, we introduced Conceptor-based Pseudo-Rehearsal (CPR), which prevents forgetting in each layer by requiring the change of upstream weights to only result in change of activation in the free subspace of this layer. In practice, this requirement is enforced by rehearsing randomly generated pseudo-inputs and their corresponding pseudo-targets in the already used subspace. In this way, learning new tasks does not overwrite the representations in the already used subspace and the mapping for previous tasks can be preserved.

We then moved on to the more challenging and less explored task-agnostic scenario of continual learning. In Chapter 4, we showed that the learning process of many real applications such as multi-agent reinforcement learning games or generative adversarial network correspond to this scenario. Due to the lack of task information and the potential conflict between different tasks, most existing continual learning methods simply could not be applied to this scenario. In order to overcome these limitations, we have proposed the What & How framework for inferring the task information and resolving the conflict between different tasks. More specifically, we showed that when the model is not informed about the task identity and the current task requires a different solution than a previous task, forgetting is simply inevitable. Therefore, we shifted our main focus from less forgetting to faster remembering. This led us to adopt meta-learning as a task-inference mechanism. At each time step, the meta-model infers the current task based on its immediate context data and selects the corresponding model to solve the current task. In this way, when the task changes, the meta model can quickly respond to the change by returning the right task-specific model. Instead of continually learning a base model, our method focuses on continually learning the meta model. From this perspective, our framework can also be viewed as a continual meta learning approach.

To further deepen our understanding of forgetting, we studied this problem from the perspective of compression in Chapter 5. This new perspective allowed us to provide a formal definition of forgetting that is task-independent and also aligned with our common interpretation of forgetting as information loss. In particular, for any data point x, we define the amount of information forgot about x caused by a change of model parameters as the increase of code length when x is encoded with the help of this model. With this formal definition, it is possible to measure forgetting precisely in number of bits. Furthermore, the perspective of compression also offered a unifying view of two main approaches to continual learning as approximations to prequential coding methods: the prior-focused approach corresponds to Bayesian mixture code, and the likelihood-focused approach corresponds to Maximum Likelihood (ML) Plug-in code. Finally we combined the predictive strategies of these two prequential coding methods and proposed a new continual learning paradigm called ML Mixture code. A Greedy recursive update rule for ML Mixture code was designed to address the limitations of prior-focused approaches.

In the last Chapter, we applied continual learning ideas to real problems occurred in the application of neuromorphic engineering. In particular, we were facing a challenging task of realizing a well-performing reservoir on an analog, unclocked, spiking, low precision, heterogeneous and slightly noisy neuromorphic microchip, very similar to the biological brain in many ways. Instead of taking the paths of manually designing the topology of such a network or training it with Back-Prop Through Time (BPTT), which were both infeasible due to the undesirable properties of the hardware, we followed the thinking of continual lifelong learning and proposed an algorithm called Reservoir Transfer to distill the knowledge from a well-understood system, namely, a leaky Echo State Network simulated in digital computer, and transferred it to the spiking network on the intricate analog device. As explained and demonstrated in Chapter 6, the Reservoir Transfer algorithm was able to circumvent all the difficulties associated with the peculiar features of the hardware: it only uses the input and output spike signals and doses not require the measurement of the internal analog values; there is no need to back-propagate errors through the discrete spikes, which are non-differentiable; the transferring process is essentially a linear regression, which only has to be performed once and can be done very efficiently; existing method for quantized linear regression is available, so the resulting weights can be ternary and thus compatible with the hardware constraint that the weights can only be ternary; since the network was trained on spike responses recorded from the chip, device mismatch is automatically taken into consideration during training; finally, the regularization applied for linear regression can prevent the weights from over-fitting the small noise in device.

7.2 LIMITATIONS

As with all machine learning algorithms, the methods introduced in this thesis have limitations. In fact, the motivation for most chapters in this thesis was to address the limitations of the settings or algorithms proposed in previous chapters.

For example, when the input space of different tasks have significant overlap, the CAB algorithm proposed in Chapter 2 will suffer from the gradient vanishing problem and will fail at learning the new task well. This limitation is addressed in Chapter 3 by the CPR algorithm. Since CAB relies on the free subspace in the input space to continually learn new tasks and the dimension of the input is fixed, CPR instead constrains the learning process based on the output space of each layer, thus leaving the entire input space available to all tasks.

The CPR method, however, has its own limitations. First, it relies on pseudo-rehearsal to maintain the functional mapping from the input space to the free output subspace, therefore it inherits the limitations of pseudo-rehearsal methods, which do not scale well with very high dimensional data, since the probability density of real data under a high dimensional Gaussian distribution is very small. There are two potential solutions to address this limitation. One way is to apply this method to a deep generative model, and replace the Gaussian samples used for rehearsal by the generated data from the model itself. This will result in a hybrid continual learning approach that combines deep generative replay and the conceptor constraints. The second solution is to introduce a regularization mechanism to the training process of the network such that the used subspace in each hidden layer corresponds to a canonical linear subspace. In this way, each dimension in the used subspace corresponds to a neuron in this layer and the constraint that any change to the pre-synaptic weights should not cause change of activation in the used subspace can be simply enforced by freezing the weights connected to the subset of neurons corresponding to the canonical subspace, thus there is no need to employ pseudo-rehearsal.

Both CAB and CPR share the same limitation that they require task boundaries to know when to update the conceptor matrices, thus they can only be applied in the task-aware scenario. To address the task-agnostic scenario, we proposed the What & How framework based on meta-learning techniques in Chapter 4. This framework relies on Bayesian online learning to combat catastrophic forgetting, therefore it shares the drawbacks of all prior-focused approaches such as the prior overhead and the large approximation gap between real posterior and simple variational posterior, as we have discussed in details in Chapter 5. Another limitation of the What & How framework is that it assumes that a fixed window of k previous observations are sufficient to infer the task information. However, this assumption might be violated if the task changes with a long temporal dependency. A straightforward fix for this problem is to use an LSTM-based meta-learner instead of an optimization-based meta-learner. However, current methods for online learning recurrent networks such as truncated BPTT or real-time recurrent learning (RTRL) [Jaeger, 2002] do not scale well with sequence length or network size. This remains a challenge for future research.

In Chapter 5, to address the limitations of prior-focused approaches, we have proposed a new prequential coding method called ML Mixture Code and designed a greedy update rule for continual learning. The main downside for this new paradigm is that, as a greedy algorithm, the approximate posterior component at each time step is optimized only with the data currently available and has to stay fixed in the future. Therefore the components optimized at earlier steps might no longer be globally optimal at a later time. A potential fix is to introduce a certain form of generative replay so that earlier components can also be adjusted later. Another downside, as we have discussed at the end of Chapter 5, is that this new paradigm also relies on a distribution approximation by a variational method, although this approximation is more plausible than the one assumed by prior-focused methods, a small approximation gap can still cause arbitrarily large increase of the loss function. In the future, it would be preferable to develop a different method that can bound the error by reducing the approximation gap.

Finally, we discuss the limitations of the Reservoir Transfer method. Although it can efficiently and successfully realize a functional reservoir on an analog neuromorphic hardware with many unconventional and undesirable properties, it assumes that we already have a well-performing teacher system available and there is an interface between the teacher system and the student system that allows them to interact with each other. Therefore, this method cannot be applied when learning has to be performed directly on the device or if we do not have a working teacher system already. Another potential problem with this method is that, since our goal is to obtain a general-purpose and task-independent reservoir, the transferring procedure matches the dynamics of the two systems based on their responses to white noise signals, which might have different distributions than those of the real signals to be processed in the future. When this difference is large, the transferred weights might over-fit to the white noise and cannot generalize to the real signals. This, however, might not be a big problem if we know in advance what data will be processed later. In such cases, we can use samples of real signal instead of white noise for the transfer process.

7.3 DISCUSSION AND FUTURE WORK

Although continual lifelong learning has become a very popular topic within machine learning in the past few years and many algorithms have been proposed to achieve this goal, these methods still perform considerably worse than the i.i.d joint training strategy on large-scale datasets in challenging scenarios such as incremental classification [Lange et al., 2021]. For this reason, for most applications, machine learning practitioners would still prefer storing all the data in the history and simply retrain the model every time new data become available. Comparing to this embarrassingly simple store-everything-and-retrain strategy, current continual learning methods usually only have demonstrated advantage in terms of memory consumption, since they do not require all the data from the past and usually assume the memory budget is bounded. However, as a Turing Award winner used to tell the author, *memory is cheap!* Indeed, with the ever-decreasing cost of information storage [McCallum, 2020a,b] and the advance of cloud computing [Shamsi, Khojaye, and Qasmi, 2013], nowadays even for the very large-scale applications, most companies in the industry have no problem to store all the data available. Furthermore, there are experimental results which suggest that when not all data can be stored, simply greedily storing as much as possible while balancing the data in each class and retraining the model at testing time will outperform most continual learning methods [Prabhu, Torr, and Dokania, 2020]. These findings plus the fact that memory is usually not expensive cast doubt on the current focus in continual learning. It seems that except for some very specialized application scenarios such as intelligent health-care [Lee and Lee, 2020], neuromorphic hardware [Indiveri and Liu, 2015] or embedded devices [Joyce and Audsley, 2016], where not all data can be stored due to privacy or limited memory resource, the simple store-everything-and-retrain strategy should be preferred over most of the current continual learning methods.

Therefore, to develop realistically useful continual learning algorithms in the future, perhaps we should relax the restriction on memory budget and instead shift its focus to address other limitations of the store-everything-and-retrain strategy: its high computational cost and inability of real-time inference. When the tasks are very different, the current retraining strategy usually still takes many epochs before it converges to a new solution, even when the retraining starts from an initialization that has been pre-trained on previous tasks [Raghu et al., 2019]. This suggests that, to gain more comparative advantage against the retraining method, practical continual learning methods should either reduce computational complexity per iteration, converge faster or require less data for retraining. Starting from this new perspective, we foresee several potential topics to be explored in the future:

First, the continual learning community should establish different benchmarks and metrics for comparing computational complexity. Most existing works on continual learning only focus on measuring forgetting but neglect the computational costs or convergence speed when evaluating their methods. Since memory is no longer a constraint in this new computation-centric paradigm, forgetting may not be a problem anymore, we should instead create tasks that require fast convergence and online processing.

Second, even if we have large enough memory to store everything we want, what we store still makes a difference. Should we store data as in the replay-based approaches or models as in the expansion-based approaches? It might not be an optimal strategy to just store the raw data, since it is usually computation-intensive to convert raw data to models and our goal is now to reduce computation cost. On the other hand, if we store models, although we can directly use them for inference, we now need to run inference on many models to select and combine their results before we can make a prediction. This will also result in substantially more computation when the number of models is large. It remains an open question which type of storage is better in terms of computation.

Third, we might want to store other forms of memory medium than raw data or models. It is unclear yet which memory medium is the best, but alternative forms do exist. For instance, hidden states in a recurrent network are neither raw data nor model parameters, yet they are usually considered as a form of memory. Another example is conceptors, which are not raw data but also not models that can be directly used for inference. However when combined with a base model, conceptors can be used to recover memory from the past [Jaeger, 2014]. Ideally, the alternative memory medium should be an intermediate form between raw data and model parameters such that they can be combined and converted to a single model and the conversion cost in terms of computation should be less than from raw data. In this way, we can always first convert the raw data into this intermediate form and whenever new data arrive, we do not have to repeat the effort spent on converting previous data to this intermediate form.

Finally, we should develop a formal theory for the trade-off between memory, computation and task performance. Although nowadays computation is generally more valuable than memory, there are still scenarios where memory is the bottleneck instead of computation, as we mentioned before. Therefore, there can be diverse application scenarios where we want to trade one performance for another one. A formal guidance for how to find the optimal compromise among these three performance dimensions is still missing for continual lifelong learning.

BIBLIOGRAPHY

Abadi, Martín et al. (2015). "TensorFlow: large-scale machine learning on heterogeneous systems." Software available from tensorflow.org.

Acar, Abbas, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti (July 2018). "A survey on homomorphic encryption schemes: theory and implementation." In: *ACM computing survey* 51.4. ISSN: 0360-0300. DOI: `10.1145/3214303`. URL: `https://doi.org/10.1145/3214303`.

Aljundi, Rahaf, Min Lin, Baptiste Goujaud, and Yoshua Bengio (2019). "Gradient based sample selection for online continual learning." arXiv: `1903.08671 [cs.LG]`.

Andrychowicz, Marcin, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas (2016). "Learning to learn by gradient descent by gradient descent." In: *Advances in neural information processing systems*, pp. 3981–3989.

Arora, Sanjeev, Rong Ge, Behnam Neyshabur, and Yi Zhang (2018). "Stronger generalization bounds for deep nets via a compression approach." arXiv: `1802.05296 [cs.LG]`.

Bagdasaryan, Eugene, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov (2020). "How to backdoor federated learning." In: *International conference on artificial intelligence and statistics*. PMLR, pp. 2938–2948.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate." arXiv: `1409.0473 [cs.CL]`.

Bakker, Bart and Tom Heskes (2003). "Task clustering and gating for bayesian multitask learning." In: *Journal of machine learning research* 4.May, pp. 83–99.

Baldi, Pierre and Roman Vershynin (2019). "The capacity of feedforward neural networks." In: *Neural networks* 116, pp. 288–311.

Bengio, Yoshua, Jérôme Louradour, Ronan Collobert, and Jason Weston (2009). "Curriculum learning." In: *International conference on machine learning*, pp. 41–48.

Birmingham, Karen, Viviana Gradinaru, Polina Anikeeva, Warren M Grill, Victor Pikov, Bryan McLaughlin, Pankaj Pasricha, Douglas Weber, Kip Ludwig, and Kristoffer Famm (2014). "Bioelectronic medicines: a research roadmap." In: *Nature reviews drug discovery* 13.6, pp. 399–400.

Blier, Léonard and Yann Ollivier (2018). "The description length of deep learning models." In: *Advances in neural information processing systems*, pp. 2216–2226.

Brette, Romain and Wulfram Gerstner (2005). "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity." In: *Journal of neurophysiology* 94.5, pp. 3637–3642.

Broderick, Tamara, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan (2013). "Streaming variational Bayes." In: *Advances in neural information processing systems*, pp. 1727–1735.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. (2020). "Language models are few-shot learners." arXiv: `2005.14165 [cs.CL]`.

Buciluă, Cristian, Rich Caruana, and Alexandru Niculescu-Mizil (2006). "Model compression." In: *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 535–541.

Burkitt, Anthony N (2006). "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input." In: *Biological cybernetics* 95.1, pp. 1–19.

Casado, Fernando E, Dylan Lema, Roberto Iglesias, Carlos V Regueiro, and Senén Barro (2020). "Concept drift detection and adaptation for robotics and mobile devices in federated and continual settings." In: *Workshop of physical agents*. Springer, pp. 79–93.

Cassandra, Anthony R., Leslie Pack Kaelbling, and Michael L. Littman (1994). "Acting optimally in partially observable stochastic domains." In: *AAAI*, pp. 1023–1028. URL: `http://www.aaai.org/Library/AAAI/1994/aaai94-157.php`.

Chaitin, Gregory (2002). "On the intelligibility of the universe and the notions of simplicity, complexity, and irreducibility." arXiv: `math/0210035 [math.HO]`.

Chaudhry, Arslan, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr (2018). "Riemannian walk for incremental learning: understanding forgetting and intransigence." In: *Proceedings of the european conference on computer vision (ECCV)*, pp. 532–547.

Chaudhry, Arslan, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny (2019). "Efficient lifelong learning with A-GEM." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=Hkf2_sC5FX`.

Che, Tong, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li (2016). "Mode regularized generative adversarial networks." arXiv: `1612.02136 [cs.LG]`.

"Continual learning Workshop NeurIPS 2018." `https://sites.google.com/view/continual2018/home`. Accessed: 2020-10-08.

Cover, Thomas M. and Joy A. Thomas (2006). *Elements of information theory (wiley series in telecommunications and signal processing)*. USA: Wiley-Interscience. ISBN: 0471241954.

Csurka, Gabriela (2017). *A comprehensive survey on domain adaptation for visual applications*. In: *Domain adaptation in computer vision applications*. Springer, pp. 1–35.

Dawid, A Philip (1984). "Present position and potential developments: some personal views statistical theory the prequential approach." In: *Journal of the royal statistical society: series A (general)* 147.2, pp. 278–290.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: pre-training of deep bidirectional transformers for language understanding." arXiv: `1810.04805 [cs.CL]`.

"Dynap-se user guide" (2017). https://docs.google.com/document/u/1/d/e/2PACX-1vQV36QRWsQl4ROfvRo7mbHb5_ZQ4Q1Qw64AkfdhuPEtIXYq1kf_ZsD3-GZkYPKqrlkOiizCq-Jjt_kD/pub?embedded=true.

Ebrahimi, Sayna, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach (2020). "Uncertainty-guided continual learning with bayesian neural networks." In: *International conference on learning representations*. URL: https://openreview.net/forum?id=HklUCCVKDB.

Farquhar, Sebastian and Yarin Gal (2019a). "A unifying bayesian view of continual learning." arXiv: 1902.06494 [stat.ML].

Farquhar, Sebastian and Yarin Gal (2019b). "Towards robust evaluations of continual learning." arXiv: 1805.09733 [stat.ML].

Fayek, Haytham M, Lawrence Cavedon, and Hong Ren Wu (2020). "Progressive learning: a deep learning framework for continual learning." In: *Neural networks* 128, pp. 345–357.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-agnostic meta-learning for fast adaptation of deep networks." In: *International conference on machine learning*. PMLR, pp. 1126–1135.

Finn, Chelsea, Aravind Rajeswaran, Sham Kakade, and Sergey Levine (2019). "Online meta-learning." In: *International conference on machine learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 1920–1930.

Flennerhag, Sebastian, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell (2020). "Meta-learning with warped gradient descent." In: *International conference on learning representations*. URL: https://openreview.net/forum?id=rkeiQlBFPB.

French, Robert M (1999). "Catastrophic forgetting in connectionist networks." In: *Trends in cognitive sciences* 3.4, pp. 128–135.

Fritzke, Bernd (1994). "A growing neural gas network learns topologies." In: *Advances in neural information processing systems* 7.

Ganguli, Surya, Dongsung Huh, and Haim Sompolinsky (2008). "Memory traces in dynamical systems." In: *Proceedings of the national academy of sciences of the united states of america* 105.48, p. 18970.

Garnelo, Marta, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami (2018). "Conditional neural processes." In: *International conference on machine learning*, pp. 1690–1699.

Geyer, Robin C., Tassilo Klein, and Moin Nabi (2018). "Differentially private federated learning: a client level perspective." arXiv: 1712.07557 [cs.CR].

Glorot, Xavier and Yoshua Bengio (May 2010). "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256.

Goldberger, Ary L, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley (2000). "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals." In: *Circulation* 101.23, e215–e220.

Golkar, Siavash, Michael Kagan, and Kyunghyun Cho (2019). "Continual learning via neural pruning." arXiv: `1903.04476 [cs.LG]`.

Goodfellow, Ian J, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio (2014a). "An empirical investigation of catastrophic forgetting in gradient-based neural networks." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=oXSw7laxwUpln`.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. `http://www.deeplearningbook.org`. MIT Press.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014b). "Generative adversarial nets." In: *Advances in neural information processing systems*, pp. 2672–2680.

Gou, Jianping, Baosheng Yu, Stephen J Maybank, and Dacheng Tao (2021). "Knowledge distillation: a survey." In: *International journal of computer vision* 129.6, pp. 1789–1819.

Grünwald, Peter D (2007). *The minimum description length principle*. The MIT Press.

Guss, William H. and Ruslan Salakhutdinov (2018). "On characterizing the capacity of neural networks using algebraic topology." arXiv: `1802.04443 [cs.LG]`.

Ha, David, Andrew M. Dai, and Quoc V. Le (2017). "Hypernetworks." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=rkpACe1lx`.

Hadaeghi, Fatemeh, Xu He, and Herbert Jaeger (2017). *Unconventional information processing systems, novel hardware: a tour d'horizon*. Tech. rep. Jacobs University Bremen. URL: `https://www.ai.rug.nl/minds/uploads/36_HadaeghiHeJaeger.pdf`.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2001). *The elements of statistical learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Hazan, Elad (2021). "Introduction to online convex optimization." arXiv: `1909.05207 [cs.LG]`.

He, Xu (2018a). "Continual learning by conceptor regularization." In: *Continual learning workshop (NeurIPS 2018)*. URL: `https://sites.google.com/view/continual2018/submissions`.

He, Xu (2018b). "Transfer learning for reservoir computing in neuromorphic hardware with low bit precision." Abstract in: *Cognitive computing: merging concepts with hardware*. URL: https://drive.google.com/open?id=1By_v5pwVEyPVFJweXlPyLDuYDeEmw9sq.

He, Xu and Herbert Jaeger (2017). "Overcoming catastrophic interference by conceptors." arXiv: 1707.04853 [cs.NE].

He, Xu and Herbert Jaeger (2018). "Overcoming catastrophic interference using conceptor-aided backpropagation." In: *International conference on learning representations*. URL: https://openreview.net/forum?id=B1al7jg0b.

He, Xu and Min Lin (2020). "Continual learning from the perspective of compression." In: *Lifelong learning workshop (ICML 2020)*. URL: https://openreview.net/forum?id=FOyk6uDpNS7.

He, Xu, Tianlin Liu, Fatemeh Hadaeghi, and Herbert Jaeger (2019a). "Reservoir transfer on analog neuromorphic hardware." In: *2019 9th international IEEE/EMBS conference on neural engineering (NER)*. IEEE, pp. 1234–1238. URL: https://ieeexplore.ieee.org/document/8716891.

He, Xu, Jakub Sygnowski, Alexandre Galashov, Andrei A. Rusu, Yee Whye Teh, and Razvan Pascanu (2019b). "Task agnostic continual learning via meta learning." arXiv: 1906.05201 [stat.ML].

He, Xu, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu (2020). "Task agnostic continual learning via meta learning." In: *Lifelong learning workshop (ICML 2020)*. URL: https://openreview.net/forum?id=AeIzVxdJgeb.

Hermans, Michiel and Benjamin Schrauwen (2010). "Memory in linear recurrent neural networks in continuous time." In: *Neural networks* 23.3, pp. 341–355.

Hernandez-Leal, Pablo, Bilal Kartal, and Matthew E Taylor (2019). "A survey and critique of multiagent deep reinforcement learning." In: *Autonomous agents and multi-agent systems* 33.6, pp. 750–797.

Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter (2017). "GANs trained by a two time-scale update rule converge to a local nash equilibrium." In: *Advances in neural information processing systems*, pp. 6626–6637.

Hinton, Geoffrey E and Drew Van Camp (1993). "Keeping the neural networks simple by minimizing the description length of the weights." In: *Proceedings of the sixth annual conference on computational learning theory*, pp. 5–13.

Hinton, Geoffrey, Oriol Vinyals, and Jeffrey Dean (2015). "Distilling the knowledge in a neural network." In: *NIPS deep learning and representation learning workshop*. URL: http://arxiv.org/abs/1503.02531.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory." In: *Neural computation* 9.8, pp. 1735–1780.

Hochreiter, Sepp, A. Steven Younger, and Peter R. Conwell (2001). "Learning to learn using gradient descent." In: *Proceedings of the international conference on artificial neural networks*. ICANN '01. London, UK, UK: Springer-Verlag, pp. 87–94. ISBN: 3-540-42486-5. URL: `http://dl.acm.org/citation.cfm?id=646258.684281`.

Hoffman, Matthew D and Matthew J Johnson (2016). "Elbo surgery: yet another way to carve up the variational evidence lower bound." In: *Workshop in advances in approximate bayesian inference*.

Hoi, Steven CH, Doyen Sahoo, Jing Lu, and Peilin Zhao (2021). "Online learning: a comprehensive survey." In: *Neurocomputing* 459, pp. 249–289.

Hospedales, Timothy, Antreas Antoniou, Paul Micaelli, and Amos Storkey (2020). "Meta-learning in neural networks: a survey." arXiv: `2004.05439 [cs.LG]`.

Indiveri, Giacomo and Shih-Chii Liu (2015). "Memory and information processing in neuromorphic systems." In: *Proceedings of the IEEE* 103.8, pp. 1379–1397.

Jaeger, H. (2007). "Echo state network." In: *Scholarpedia* 2.9. revision #189893, p. 2330. DOI: `10.4249/scholarpedia.2330`.

Jaeger, Herbert (2001). *The "echo state" approach to analysing and training recurrent neural networks-with an erratum note*. Tech. rep. 148. German National Research Center for Information Technology. URL: `https://www.ai.rug.nl/minds/uploads/EchoStatesTechRepErratum.pdf`.

Jaeger, Herbert (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network"approach*. Tech. rep. 159. German National Research Center for Information Technology. URL: `https://www.ai.rug.nl/minds/uploads/ESNTutorialRev.pdf`.

Jaeger, Herbert (2014). *Controlling recurrent neural networks by conceptors*. Tech. rep. 31. `https://arxiv.org/abs/1403.3369`. Jacobs University Bremen.

Jaeger, Herbert (2017). *Using conceptors to manage neural long-term memories for temporal patterns*. Tech. rep. 13, pp. 1–43. URL: `http://jmlr.org/papers/v18/15-449.html`.

Jaeger, Herbert and Harald Haas (2004). "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication." In: *Science* 304.5667, pp. 78–80.

Javed, Khurram and Martha White (2019). "Meta-learning representations for continual learning." In: *Advances in neural information processing systems*, pp. 1818–1828.

Joyce, Russell and Neil Audsley (2016). "Exploring storage bottlenecks in Linux-based embedded systems." In: *ACM SIGBED review* 13.1, pp. 54–59.

Kemker, Ronald, Angelina Abitino, Marc McClure, and Christopher Kanan (2017). "Measuring catastrophic forgetting in neural networks." In: *Computing research repository* abs/1708.02072. `http://arxiv.org/abs/1708.02072`.

Kessler, Samuel, Vu Nguyen, Stefan Zohren, and Stephen Roberts (2019). "Indian buffet neural networks for continual learning." In: *4th workshop on bayesian deep learning (NeurIPS 2019)*.

Khan, Samee U, Albert Y Zomaya, and Assad Abbas (2017). *Handbook of large-scale distributed computing in smart healthcare*. Springer.

Kim, Miran, Yongsoo Song, Shuang Wang, Yuhou Xia, Xiaoqian Jiang, et al. (2018). "Secure logistic regression based on homomorphic encryption: design and evaluation." In: *Jmir medical informatics* 6.2, e8805.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: a method for stochastic optimization." arXiv: `1412.6980 [cs.LG]`.

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational Bayes." arXiv: `1312.6114 [stat.ML]`.

Kirkpatrick, James et al. (2017). "Overcoming catastrophic forgetting in neural networks." In: *Proceedings of the national academy of sciences* 114.13, p. 3521.

Koch, Gregory, Richard Zemel, Ruslan Salakhutdinov, et al. (2015). "Siamese neural networks for one-shot image recognition." In: *ICML deep learning workshop*.

Kolmogorov, Andrei Nikolaevich (1968). "Three approaches to the quantitative definition of information." In: *International journal of computer mathematics* 2.1-4, pp. 157–168.

Konečný, Jakub, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon (2017). "Federated learning: strategies for improving communication efficiency." arXiv: `1610.05492 [cs.LG]`.

Krizhevsky, Alex (2009). "Learning multiple layers of features from tiny images." In: *Master's thesis, University of Toronto*, pp. 32–33. URL: `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*, pp. 1097–1105.

Krogh, Anders and John A Hertz (1992). "A simple weight decay can improve generalization." In: *Advances in neural information processing systems*, pp. 950–957.

Krueger, David, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville (2017). "Bayesian hypernetworks." arXiv: `1710.04759 [stat.ML]`.

Krueger, Kai A and Peter Dayan (2009). "Flexible shaping: how learning in small steps helps." In: *Cognition* 110.3, pp. 380–394.

Kumar, Abhishek, Sunabha Chatterjee, and Piyush Rai (2019). "Nonparametric bayesian structure adaptation for continual learning." In: *CoRR* abs/1912.03624. URL: `http://arxiv.org/abs/1912.03624`.

Kumar, Abhishek and Hal Daume III au2 (2012). "Learning task grouping and overlap in multi-task learning." arXiv: `1206.6417 [cs.LG]`.

Kurle, Richard, Botond Cseke, Alexej Klushyn, Patrick van der Smagt, and Stephan Günnemann (2020). "Continual learning with bayesian neural networks for non-stationary data." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=SJlsFpVtDB`.

Lake, Brenden M, Ruslan Salakhutdinov, and Joshua B Tenenbaum (2015). “Human-level concept learning through probabilistic program induction.” In: *Science* 350.6266, pp. 1332–1338.

Lange, Matthias De, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars (2021). “A continual learning survey: defying forgetting in classification tasks.” arXiv: 1909.08383 [cs.CV].

LeCun, Yann, Corinna Cortes, and Christopher JC Burges (1998). “The MNIST database of handwritten digits.” URL: http://yann.lecun.com/exdb/mnist/.

Lee, Cecilia S and Aaron Y Lee (2020). “Clinical applications of continual learning machine learning.” In: *The lancet digital health* 2.6, e279–e281.

Lee, Hae Beom, Hayeon Lee, Donghyun Na, Saehoon Kim, Minseop Park, Eunho Yang, and Sung Ju Hwang (2020a). “Learning to balance: bayesian meta-learning for imbalanced and out-of-distribution tasks.” In: *International conference on learning representations*. URL: https://openreview.net/forum?id=rkeZIJBYvr.

Lee, Sang-Woo, Jin-Hwa Kim, JungWoo Ha, and Byoung-Tak Zhang (2017). “Overcoming catastrophic forgetting by incremental moment matching.” In: *Computing research repository* abs/1703.08475. http://arxiv.org/abs/1703.08475.

Lee, Soochan, Junsoo Ha, Dongsu Zhang, and Gunhee Kim (2020b). “A neural dirichlet process mixture model for task-free continual learning.” In: *International conference on learning representations*. URL: https://openreview.net/forum?id=SJxSOJStPr.

Lesort, Timothée, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Andrei Stoian, and David Filliat (2019). “Generative models from the perspective of continual learning.” In: *2019 international joint conference on neural networks (IJCNN)*. IEEE, pp. 1–8.

Li, D, Y Yang, Yi-Zhe Song, and TM Hospedales (2018). “Learning to generalize: meta-learning for domain generalization.” In: *Proceedings of the thirty-second AAAI conference on artificial intelligence (AAAI 2018)*. AAAI press, pp. 3490–3497.

Li, Qinbin, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He (2021). “A survey on federated learning systems: vision, hype and reality for data privacy and protection.” arXiv: 1907.09693 [cs.LG].

Li, Zhenguo, Fengwei Zhou, Fei Chen, and Hang Li (2017). “Meta-sgd: learning to learn quickly for few-shot learning.” arXiv: 1707.09835 [cs.LG].

Li, Zhizhong and Derek Hoiem (2016). “Learning without forgetting.” In: *European conference on computer vision*. Springer, pp. 614–629.

Li, Zhizhong and Derek Hoiem (2017). “Learning without forgetting.” In: *IEEE transactions on pattern analysis and machine intelligence* 40.12, pp. 2935–2947.

Liang, Kevin J, Chunyuan Li, Guoyin Wang, and Lawrence Carin (2019). “Generative adversarial network training is a continual learning problem.” URL: https://openreview.net/forum?id=SJzuHiA9tQ.

Liu, Tianlin (2019). "Harnessing slow dynamics in neuromorphic computation." arXiv: 1905.12116 [cs.LG].

Lopez-Paz, David et al. (2017). "Gradient episodic memory for continual learning." In: *Advances in neural information processing systems*, pp. 6470–6479.

Lukoševičius, Mantas (2012). *A practical guide to applying echo state networks*. In: *Neural networks: tricks of the trade*. Springer, pp. 659–686.

Luo, Sha, Hamidreza Kasaei, and Lambert Schomaker (2020). "Accelerating reinforcement learning for reaching using continuous curriculum learning." In: *2020 international joint conference on neural networks (ijcnn)*, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9207427.

Maass, Wolfgang, Thomas Natschläger, and Henry Markram (2002). "Real-time computing without stable states: a new framework for neural computation based on perturbations." In: *Neural computation* 14.11, pp. 2531–2560.

Mallya, Arun and Svetlana Lazebnik (2018). "Packnet: adding multiple tasks to a single network by iterative pruning." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7765–7773.

Masse, Nicolas Y., Gregory D. Grant, and David J. Freedman (2018). "Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization." In: *Proceedings of the national academy of sciences* 115.44, E10467–E10475. URL: https://www.pnas.org/content/115/44/E10467.

McCallum, John C. (2020a). "Disk drive prices 1955+." https://jcmit.net/diskprice.htm. Accessed: 2020-10-25.

McCallum, John C. (2020b). "Flash memory and SSD prices." https://jcmit.net/flashprice.htm. Accessed: 2020-10-25.

McCloskey, Michael and Neal J Cohen (1989). "Catastrophic interference in connectionist networks: the sequential learning problem." In: *Psychology of learning and motivation* 24, pp. 109–165.

Mehta, Nikhil, Kevin J Liang, Vinay K Verma, and Lawrence Carin (2021). "Bayesian nonparametric weight factorization for continual learning." arXiv: 2004.10098 [cs.LG].

Melis, Luca, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov (2019). "Exploiting unintended feature leakage in collaborative learning." In: *2019 IEEE symposium on security and privacy (SP)*. IEEE, pp. 691–706.

Mermillod, Martial, Aurélia Bugaiska, and Patrick Bonin (2013). "The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects." In: *Frontiers in psychology* 4, p. 504.

Metz, Luke, Ben Poole, David Pfau, and Jascha Sohl-Dickstein (2016). "Unrolled generative adversarial networks." arXiv: 1611.02163 [cs.LG].

Milan, Kieran, Joel Veness, James Kirkpatrick, Michael Bowling, Anna Koop, and Demis Hassabis (2016). "The forget-me-not process." In: *Advances in neural information processing systems*, pp. 3702–3710.

Minka, Thomas P, Rongjing Xiang, and Yuan Qi (2009). "Virtual vector machine for bayesian online classification." In: *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pp. 411–418.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, pp. 529–533.

Mohassel, Payman and Yupeng Zhang (2017). "SecureML: a system for scalable privacy-preserving machine learning." In: *2017 IEEE symposium on security and privacy (SP)*, pp. 19–38. DOI: 10.1109/SP.2017.12.

Monahan, George E (1982). "State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms." In: *Management science* 28.1, pp. 1–16.

Moody, George B and Roger G Mark (2001). "The impact of the mit-bih arrhythmia database." In: *IEEE engineering in medicine and biology magazine* 20.3, pp. 45–50.

Moradi, Saber, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri (2017). "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)." In: *IEEE transactions on biomedical circuits and systems* 12.1, pp. 106–122.

Mundt, Martin, Yong Won Hong, Iuliia Pliushch, and Visvanathan Ramesh (2020). "A wholistic view of continual learning with deep neural networks: forgotten lessons and the bridge to active and open world learning." arXiv: 2009.01797 [cs.LG].

Nagabandi, Anusha, Chelsea Finn, and Sergey Levine (2019). "Deep online learning via meta-learning: continual adaptation for model-based RL." In: *International conference on learning representations*. URL: https://openreview.net/forum?id=HyxAfnA5tm.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines." In: *International conference on machine learning*, pp. 807–814.

Neal, Radford M (2012). *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media.

Neyshabur, Behnam, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro (2018). "Towards understanding the role of over-parametrization in generalization of neural networks." arXiv: 1805.12076 [cs.LG].

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

Nichol, Alex, Joshua Achiam, and John Schulman (2018). "On first-order meta-learning algorithms." arXiv: 1803.02999 [cs.LG].

Nicola, Wilten and Claudia Clopath (2017). "Supervised learning in spiking neural networks with force training." In: *Nature communications* 8.1, pp. 1–15.

Ollivier, Yann (2014). "Auto-encoders: reconstruction versus compression." arXiv: 1403.7752 [cs.NE].

Opper, Manfred (1998). *A bayesian approach to on-line learning*. In: *On-line learning in neural networks*. Cambridge University Press Cambridge, pp. 363–378.

Oswald, Johannes von, Christian Henning, João Sacramento, and Benjamin F. Grewe (2020). "Continual learning with hypernetworks." In: *International conference on learning representations*. URL: https://openreview.net/forum?id=SJgwNerKvB.

Parisi, German I, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter (2019). "Continual lifelong learning with neural networks: a review." In: *Neural networks* 113, pp. 54–71.

Pomponi, Jary, Simone Scardapane, and Aurelio Uncini (2021). "Pseudo-rehearsal for continual learning with normalizing flows." arXiv: 2007.02443 [stat.ML].

Prabhu, Ameya, Philip Torr, and Puneet Dokania (2020). "GDumb: a simple approach that questions our progress in continual learning." In: *The european conference on computer vision (ECCV)*.

Qiao, Ning, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri (2015). "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses." In: *Frontiers in neuroscience* 9, p. 141.

Qureshi, A. H., Y. Miao, and Michael C. Yip (2020). "Active continual learning for planning and navigation." In: *ICML 2020 workshop on real world experiment design and active learning*.

Radford, Alec, Luke Metz, and Soumith Chintala (2016). "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv: 1511.06434 [cs.LG].

Raghu, Maithra, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio (2019). "Transfusion: understanding transfer learning for medical imaging." In: *Advances in neural information processing systems*, pp. 3347–3357.

Rao, Dushyant, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell (2019). "Continual unsupervised representation learning." In: *Advances in neural information processing systems*, pp. 7645–7655.

Ratcliff, Roger (1990). "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions." In: *Psychological review* 97.2, pp. 285–308.

Ravi, Sachin and H. Larochelle (2017). "Optimization as a model for few-shot learning." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=rJY0-Kcll`.

Rebuffi, Sylvestre-Alvise, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert (2017). "iCaRL: incremental classifier and representation learning." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2001–2010.

Richardson, Eitan and Yair Weiss (2018). "On GANs and GMMs." In: *Advances in neural information processing systems*, pp. 5847–5858.

Riemer, Matthew, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro (2019). "Learning to learn without forgetting by maximizing transfer and minimizing interference." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=B1gTShAct7`.

Rifkin, Ryan and Aldebaro Klautau (2004). "In defense of one-vs-all classification." In: *Journal of Machine Learning Research* 5, pp. 101–141.

Ring, Mark B (1998). *Child: a first step towards continual learning*. In: *Learning to learn*. Springer, pp. 261–292.

Ritter, Hippolyt, Aleksandar Botev, and David Barber (2018). "Online structured laplace approximations for overcoming catastrophic forgetting." In: *Advances in neural information processing systems*, pp. 3738–3748.

Robins, Anthony (1995). "Catastrophic forgetting, rehearsal and pseudorehearsal." In: *Connection science* 7.2, pp. 123–146.

Rolnick, David, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne (2019). "Experience replay for continual learning." In: *Advances in neural information processing systems*, pp. 350–360.

Ruder, Sebastian (2017). "An overview of gradient descent optimization algorithms." arXiv: `1609.04747 [cs.LG]`.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors." In: *Nature* 323, pp. 533–535.

Rusu, Andrei A., Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell (2016). "Progressive neural networks." arXiv: `1606.04671 [cs.LG]`.

Rusu, Andrei A., Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell (2019). "Meta-learning with latent embedding optimization." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=BJgklhAcK7`.

Ruvolo, Paul and Eric Eaton (2013). "ELLA: an efficient lifelong learning algorithm." In: *International conference on machine learning*, pp. 507–515.

Sabatelli, Matthia (2022). *Contributions to deep transfer learning: from supervised to reinforcement learning*. PhD thesis. ULiège - Université de Liège, Liège, Belgium.

Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen (2016). "Improved techniques for training GANs." In: *Advances in neural information processing systems*, pp. 2234–2242.

Santoro, Adam, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap (2016). "Meta-learning with memory-augmented neural networks." In: *International conference on machine learning*, pp. 1842–1850.

Schaul, Tom, Hado van Hasselt, Joseph Modayil, Martha White, Adam White, Pierre-Luc Bacon, Jean Harb, Shibl Mourad, Marc Bellemare, and Doina Precup (2018). "The Barbados 2018 list of open issues in continual learning." arXiv: `1811.07004 [cs.AI]`.

Schomaker, Lambert (Nov. 2020). *Lifelong learning for text retrieval and recognition in historical handwritten document collections*. In: *Series in machine perception and artificial intelligence*. WORLD SCIENTIFIC, pp. 221–248. DOI: `10.1142/9789811203244_0012`. URL: `https://doi.org/10.1142/9789811203244_0012`.

Schwarz, Jonathan, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell (2018). "Progress & compress: a scalable framework for continual learning." In: *International conference on machine learning*, pp. 4535–4544.

Seff, Ari (2017). "Implementation of "overcoming catastrophic forgetting in neural networks" in tensorflow." In: *Github repository*. `https://github.com/ariseff/overcoming-catastrophic`.

Serra, Joan, Didac Suris, Marius Miron, and Alexandros Karatzoglou (July 2018). "Overcoming catastrophic forgetting with hard attention to the task." In: *International conference on machine learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4548–4557. URL: `https://proceedings.mlr.press/v80/serra18a.html`.

Settles, Burr (2009). *Active learning literature survey*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences. URL: `https://minds.wisconsin.edu/bitstream/handle/1793/60660/TR1648.pdf`.

Shamsi, Jawwad, Muhammad Ali Khojaye, and Mohammad Ali Qasmi (2013). "Data-intensive cloud computing: requirements, expectations, challenges, and solutions." In: *Journal of grid computing* 11.2, pp. 281–310.

Shannon, Claude Elwood (2001). "A mathematical theory of communication." In: *ACM SIGMOBILE mobile computing and communications review* 5.1, pp. 3–55.

Shaw, Albert, Wei Wei, Weiyang Liu, Le Song, and Bo Dai (2019). "Meta architecture search." In: *Advances in neural information processing systems*, pp. 11227–11237.

Shin, Hanul, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim (2017). "Continual learning with deep generative replay." In: *Advances in neural information processing systems*, pp. 2990–2999.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587, pp. 484–489.

Smith, Virginia, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S. Talwalkar (2017). "Federated multi-task learning." In: *Advances in neural information processing systems*, pp. 4427–4437. URL: `http://papers.nips.cc/paper/7029-federated-multi-task-learning`.

Solomonoff, Ray J (1964). "A formal theory of inductive inference. part I." In: *Information and control* 7.1, pp. 1–22.

Sontag, Eduardo D. (1998). "VC dimension of neural networks." In: *Neural networks and machine learning*. Springer, pp. 69–95.

Srivastava, Rupesh K, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber (2013). "Compete to compute." In: *Advances in neural information processing systems*. `http://papers.nips.cc/paper/5059-compete-to-compute.pdf`, pp. 2310–2318.

Stark, Jaroslav (1999). "Delay embeddings for forced systems. I. deterministic forcing." In: *Journal of nonlinear science* 9.3, pp. 255–332.

Stark, Jaroslav, David S Broomhead, ME Davies, and J Huke (2003). "Delay embeddings for forced systems. II. stochastic forcing." In: *Journal of nonlinear science* 13.6, pp. 519–577.

Stimberg, Marcel, Romain Brette, and Dan FM Goodman (Aug. 2019). "Brian 2, an intuitive and efficient neural simulator." In: *eLife* 8. Ed. by Frances K Skinner, e47314. ISSN: 2050-084X. DOI: `10.7554/eLife.47314`.

Swaroop, Siddharth, Cuong V. Nguyen, Thang D. Bui, and Richard E. Turner (2019). "Improving and understanding variational continual learning." arXiv: `1905.02099 [stat.ML]`.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Takesian, Anne E and Takao K Hensch (2013). *Balancing plasticity/stability across brain development*. In: *Progress in brain research*. Vol. 207. Elsevier, pp. 3–34.

Tesauro, Gerald et al. (1995). "Temporal difference learning and td-gammon." In: *Communications of the ACM* 38.3, pp. 58–68.

Thanh-Tung, Hoang and Truyen Tran (2020). "Catastrophic forgetting and mode collapse in GANs." In: *2020 international joint conference on neural networks (ijcnn)*, pp. 1–10. DOI: `10.1109/IJCNN48605.2020.9207181`.

Thrun, Sebastian (1998). *Lifelong learning algorithms*. In: *Learning to learn*. Springer, pp. 181–209.

Tishby, Naftali and Noga Zaslavsky (2015). "Deep learning and the information bottleneck principle." In: *2015 IEEE information theory workshop (ITW)*. IEEE, pp. 1–5.

Tolstikhin, Ilya, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf (2019). "Wasserstein auto-encoders." arXiv: `1711.01558 [stat.ML]`.

Tsai, Wei-Yu, Davis R Barch, Andrew S Cassidy, Michael V DeBole, Alexander Andreopoulos, Bryan L Jackson, Myron D Flickner, John V Arthur, Dharmendra S Modha, John Sampson, et al. (2017). "Always-on speech recognition using truenorth, a reconfigurable, neurosynaptic processor." In: *IEEE transactions on computers* 66.6, pp. 996–1007.

Vanschoren, Joaquin (2018). "Meta-learning: a survey." arXiv: `1810.03548 [cs.LG]`.

Ven, Gido M. van de and Andreas S. Tolias (2019). "Three scenarios for continual learning." arXiv: `1904.07734 [cs.LG]`.

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. (2019). "Grandmaster level in starcraft II using multi-agent reinforcement learning." In: *Nature* 575.7782, pp. 350–354.

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. (2016). "Matching networks for one shot learning." In: *Advances in neural information processing systems*, pp. 3630–3638.

Wilson, Garrett and Diane J Cook (2020). "A survey of unsupervised deep domain adaptation." In: *ACM transactions on intelligent systems and technology (TIST)* 11.5, pp. 1–46.

Wu, Chenshen, Luis Herranz, Xialei Liu, yaxing wang, Joost van de Weijer, and Bogdan Raducanu (2018). *Memory replay GANs: learning to generate new categories without forgetting*. In: *Advances in neural information processing systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 5962–5972.

Yang, Qiang, Yang Liu, Tianjian Chen, and Yongxin Tong (2019). "Federated machine learning: concept and applications." arXiv: `1902.04885 [cs.AI]`.

Yoon, Jaehong, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang (2021). "Federated continual learning with weighted inter-client transfer." In: *International conference on machine learning*. PMLR, pp. 12073–12086.

Yoon, Jaehong, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang (2018). "Lifelong learning with dynamically expandable networks." arXiv: `1708.01547 [cs.LG]`.

Yosinski, Jason, Jeff Clune, Yoshua Bengio, and Hod Lipson (2014). "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*, pp. 3320–3328.

Zenke, Friedemann, Ben Poole, and Surya Ganguli (2017). "Continual learning through synaptic intelligence." In: *International conference on machine learning*. JMLR. org, pp. 3987–3995.

Zeno, Chen, Itay Golan, Elad Hoffer, and Daniel Soudry (2018). "Bayesian gradient descent: online variational Bayes learning with increased robustness to catastrophic forgetting and weight pruning." In: *CoRR* abs/1803.10123. URL: `http://arxiv.org/abs/1803.10123`.

Zhang, Yu and Qiang Yang (2021). "A survey on multi-task learning." arXiv: `1707.08114 [cs.LG]`.

Zhao, Yue, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra (2018). "Federated learning with non-IID data." arXiv: `1806.00582 [cs.LG]`.

Zhu, Chenzhuo, Song Han, Huizi Mao, and William J Dally (2017). "Trained ternary quantization." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=S1_pAu9xl&noteId=S1_pAu9xl`.

Zhuang, Fuzhen, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He (2020). "A comprehensive survey on transfer learning." In: *Proceedings of the IEEE* 109.1, pp. 43–76.

Zintgraf, Luisa, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson (June 2019). "Fast context adaptation via meta-learning." In: *International conference on machine learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 7693–7702. URL: `https://proceedings.mlr.press/v97/zintgraf19a.html`.

PUBLICATIONS

The following is a list of publications I worked on during my Ph.D. studies and a description of my contributions to each of them.

Hadaeghi, Fatemeh, Xu He, and Herbert Jaeger (2017). *Unconventional information processing systems, novel hardware: a tour d'horizon*. Tech. rep. Jacobs University Bremen. URL: `https://www.ai.rug.nl/minds/uploads/36_HadaeghiHeJaeger.pdf`.

He, Xu (2018a). "Continual learning by conceptor regularization." In: *Continual learning workshop (NeurIPS 2018)*. URL: `https://sites.google.com/view/continual2018/submissions`.

He, Xu (2018b). "Transfer learning for reservoir computing in neuromorphic hardware with low bit precision." Abstract in: *Cognitive computing: merging concepts with hardware*. URL: `https://drive.google.com/open?id=1By_v5pwVEyPVFJweXlPyLDuYDeEmw9sq`.

He, Xu and Herbert Jaeger (2017). "Overcoming catastrophic interference by conceptors." arXiv: `1707.04853 [cs.NE]`.

He, Xu and Herbert Jaeger (2018). "Overcoming catastrophic interference using conceptor-aided backpropagation." In: *International conference on learning representations*. URL: `https://openreview.net/forum?id=B1al7jg0b`.

He, Xu and Min Lin (2020). "Continual learning from the perspective of compression." In: *Lifelong learning workshop (ICML 2020)*. URL: `https://openreview.net/forum?id=FOyk6uDpNS7`.

He, Xu, Tianlin Liu, Fatemeh Hadaeghi, and Herbert Jaeger (2019a). "Reservoir transfer on analog neuromorphic hardware." In: *2019 9th international IEEE/EMBS conference on neural engineering (NER)*. IEEE, pp. 1234–1238. URL: `https://ieeexplore.ieee.org/document/8716891`.

He, Xu, Jakub Sygnowski, Alexandre Galashov, Andrei A. Rusu, Yee Whye Teh, and Razvan Pascanu (2019b). "Task agnostic continual learning via meta learning." arXiv: `1906.05201 [stat.ML]`.

The author's contribution to the technical report [Hadaeghi, He, and Jaeger, 2017] is the first draft of Section 2.5 and Section 4.3. In the case of He and Jaeger [2017, 2018], the first author designed the Conceptor-Aided-Backprop algorithm for the multi-layer case, implemented all of the experiments and wrote the majority of the paper. For the Reservoir Transfer paper [He et al., 2019a], the first author designed the algorithm, implemented all of the simulated experiments and part of the on-device experiments. The first author also wrote the entire section on method and part of the section on experiments. In the case of the continual meta learning paper [He et al., 2019b], the majority of the ideas, experiments and writing originated from the first author. For the paper on continual learning and compression [He and Lin, 2020], both authors contributed equally to all material. Finally, He [2018a,b] are both single-authored papers.

SAMENVATTING

Agenten met kunstmatige intelligentie horen gedurende hun gehele levensduur kennis moeten vergaren en vaardigheden moeten op pakken. Continual learning verwijst naar de scenario's waarin een machinaal lerend systeem eerder verkregen vaardigheden kan behouden en benutten terwijl het nieuwe leert. Wanneer connectionistische modellen getraind worden op een reeks van taken, vergeten ze echter doorgaans eerder vergaarde kennis nadat hun parameters zijn aangepast voor een nieuwe taak. Dit beruchte probleem, bekend als catastrophic forgetting, vormt een serieuze uitdaging voor voortdurend levenslang leren. In deze dissertatie leggen wij nieuwe oplossingen voor, voor het probleem van catastrophic forgetting, continual learning en transfer learning.

- In hoofdstuk 2 stellen wij een variant van het backpropagation algoritme voor, Conceptor-Aided Backprop (CAB) [He and Jaeger, 2017, 2018], waar gradiënten worden afgeschermd tegen de degradatie van eerder geleerde taken. Conceptors vinden hun oorsprong in reservoir computing, waar eerder is aangetoond dat ze catastrophic forgetting tegen kunnen gaan. CAB breidt deze resultaten uit naar diepe feedforward netwerken. In het bijzonder passen wij in elke laag van het diepe netwerk een conceptor toe om de voorheen gebruikte lineaire deelruimte te identificeren, en de gradiënten te projecteren op de vrije deelruimte voor toekomstig leren.

- We laten zien dat wanneer de invoerruimten van twee taken een grote mate van overlap overlappen hebben, het toepassen van CAB zal leiden tot onverzettelijkheid, oftewel het onvermogen van een netwerk om nieuwe kennis te leren. Wij analyseren de achterliggende reden en stellen in hoofdstuk 3 een andere CL-methode voor, *Conceptor Pseudo-Rehearsal* (CPR) [He, 2018a], om met deze beperking om te gaan. In plaats van de parameter updates te projecteren op de vrije deelruimte in de presynaptische laag, beperkt CPR ze door de vrije deelruimte in de post-synaptische laag. Nieuwe activeringen als gevolg van aanpassingen van de parameters verschijnen alleen in de vrije deelruimte. Dientengevolge interfereren deze activeringen niet met oude taken. Daarenboven kunnen nuttige features van de oude taken worden hergebruikt door de nieuwe taken, wat een voorwaartse overdracht tussen taken mogelijk maakt.

- De meeste benaderingen voor continual learning gaan er impliciet van uit dat er een multitaak oplossing voor meerdere taken bestaat. In hoofdstuk 4 onderbouwen en bespreken wij realistische scenario's, zoals multiagent spellen, waarin deze

veronderstelling niet opgaat. Wij beargumenteren dat de traditionele metriek van zero-shot onthouden niet geschikt is in dergelijke omstandigheden en wij richten ons op de snelheid van het onthouden van voorgaande taken, geïnspireerd door de meta-leren literatuur. Wij stellen het *What and How* kader voor [He et al., 2020] om dit geval aan te pakken, waarin een onderscheid gemaakt wordt tussen welke taak momenteel wordt opgelost en hoe de taak opgelost moet worden. Bij elke stap voert het *What* algoritme taakinferentie uit, waardoor ons kader kan werken zonder taakgrenzen. Het *How* algoritme is geconditioneerd op de afgeleide taak, waardoor taakspecifiek gedrag mogelijk wordt, en de aanname van een multitaak oplossing wordt versoepeld. Vanuit het perspectief van meta-leren kan ons kader omgaan met een sequentiële presentatie van taken, in plaats van toegang te hebben tot de verdeling van alle taken. Wij valideren de doeltreffendheid van onze aanpak empirisch en passen deze toe om generative adversarial networks (GAN) te trainen.

- In hoofdstuk 5 bestuderen wij catastrophic forgetting vanuit het perspectief van de informatietheorie en definiëren wij vergeten als het toenemen van de beschrijvingslengte van eerdere data wanneer deze worden gecomprimeerd met een aaneenvolgend geleerd model [He and Lin, 2020]. Ook tonen wij aan dat methodes voor voortdurend lerend op basis van variationele posterior benadering en generative replay kunnen worden beschouwd als benaderingen van twee prequentiële coderingsmethoden in compressie, namelijk de Bayesiaanse mixture codes en grootste aannemelijkheid (GA) plug-in code. Wij vergelijken deze benaderingen in termen van zowel compressie als vergeten en bestuderen empirisch de oorzaken die de prestaties van benaderingen die gebaseerd zijn op variationele posterior benadering beperken. Om deze beperkingen aan te pakken, stellen wij een nieuwe methode voor continu leren voor, genaamd *GA Mixture Code*, die GA plug-in en Bayesiaanse mixture codes combineert

- Als laatste richten wij ons op het voorwaartse overdracht aspect van continu leren en passen dit toe op het probleem van neuromorphic engineering in hoofdstuk 6 [He, 2018b; He et al., 2019a]. Analoge, niet-geklokte, spikeende neuromorphic microchips bieden nieuwe perspectieven voor implanteerbare of draagbare biosensoren en bio-controllers, vanwege hun lage energieverbruik en warmteafvoer. De uitdagingen vanuit rekenkundig oogpunt zijn echter formidabel. Wij geven de hoofdlijnen van onze oplossingen om het reservoir computing paradigma op dergelijke hardware te realiseren en pakken de gecombineerde problemen van lage bit resolutie, apparaat mismatch, benaderende neuronmodellen en tijdschaal mismatch aan. De belangrijkste bijdrage is een berekeningsschema, *Reservoir Transfer*, dat ons in staat stelt de dynamische eigenschappen van een goed presterend neuraal netwerk dat is geoptimaliseerd op een digitale computer, over te brengen op neuromorphic hardware die de bovengenoemde problematische eigenschappen vertoont. Wij presenteren een casus van de implementatie van een ECG-hartslagdetector om de voorgestelde methode te demonstreren.

ACKNOWLEDGMENTS

Although every PhD experience is unique, I believe mine was particularly unconventional. I started at the Jacobs University Bremen in Germany but eventually moved with my supervisor to the University of Groningen in the Netherlands. In between, I interned at the University of Bern, the Max Planck Institute for Mathematics in the Sciences, DeepMind, ETH Zürich, and Mila. To add more twists to this story, a pandemic started when I began writing this dissertation. Looking back at this unusual journey, I feel very lucky to have been helped by so many people, to whom I would like to express my sincere gratitude here.

I am most grateful to my thesis advisor, Herbert Jaeger, without whom I would not have even considered a career as a scientist. It was Herbert who first introduced me to the wonderful world of AI through his enlightening lectures and reading group meetings during my undergraduate years at Jacobs. His research showed me how machine learning can blend the rigor of a mathematician and the creativity of an artist. Herbert provided more guidance, freedom, support, inspiration, and patience than I could ever hope for. He has been and will always remain my role model as a scientist, a mentor and a man. Through Herbert, I understood how one can be rigorous yet humorous, flexible yet principled, extraordinarily knowledgeable yet truly humble. Because of him, I am able to see not only the trees but also the forest, both in research and in life.

It would not have been possible to start my PhD at Jacobs without the support of Michael Kohlhase, who funded and co-supervised the first year of my PhD until he moved to Erlangen. Michael has an admirable personality and deep insight into Mathematics and Logic. If Herbert showed me the vastness of a forest, it was Michael who helped me see the grandiosity of a cathedral. His grand vision for mathematical knowledge management and automated theorem proving kindled in me a passion for bridging learning and reasoning.

I would like to thank my MINDS comrades, SeaMean and Tianlin, for fighting alongside me against the beast of analog dynamics. For that matter, I must also thank our collaborators at INI: Giacomo, Roberto, Federico, Ole, and Karsten. Without their help, we would never have been able to crack open and peek into the most enigmatic Blue Box.

During my time in Bern, where Einstein developed the theory of relativity, I felt emboldened to think about big questions such as how the brain works. More encouraging than Einstein's story was Walter Senn, who opened the door to the world of spikes and dendrites for me. Walter's group was full of friendly and supportive people: Chang, Dominik, Elena, Erik, Francesco, João, Kristin, Martin, and Pascal, who made sure my time in Switzerland was not only about neurons but also about the Aare and UEFA Euro games. I especially thank Erik for offering me a month-long experience as a resident of the lovely town of Thun.

I want to thank the Bernstein Network and the Volkswagen Foundation for the SmartStart fellowship, through which I had the chance to meet many brilliant young researchers and top scientists in neuroscience all over Germany. I was lucky to have Jürgen Jost, the director of MPI MiS and a real mathemagician, as my SmartStart mentor. Jürgen kindly invited me for a lab rotation at perhaps the best institute for applied mathematicians in Germany, where I was exposed to all kinds of exotic, tropical, and dynamical maths. I had many interesting discussions with his PhD student, Pau, who often changed my way of thinking from time to frequency.

I owe a debt of gratitude to Razvan Pascanu for being my guardian angel throughout my internship at DeepMind. He always knew the right spell to cast to fix my deep models. Whenever I felt lost in the murky waters of deep learning, Razvan was there to throw me a lifeline. A big shoutout also goes to Jakub, Alex, Andrei, and Yee Whye for their valuable contributions to my project. I couldn't have done it without their wizardry. I also want to thank Francesco, Dushyant, Jonathan, Raia, and Demis for sharing their mystical insights into the future of continual learning.

I would like to thank Benjamin for inviting me to Switzerland for a research visit at ETH despite all the difficulties, where I had many stimulating discussions with his group members João, Christian, and Johannes on online learning and hypernetworks. It was such a pleasant experience working with them.

I want to thank Yoshua Bengio for offering me an internship at Mila when I needed one the most. Yoshua's constructive criticism towards existing approaches to continual learning helped reshape my view of this field. I was fortunate to have had my close friend Anca in Montreal, who provided emergency accommodation on short notice, and her ongoing support during my stay there made the Canadian winter much warmer. I also felt really lucky to have met Min Lin at Mila. Through our countless brainstorming sessions, we must have invented all possible methods to combat forgetting, but at the same time, understood exactly why they were doomed to fail. If anyone understands my love and hate towards continual learning, it is Min.

During my time in Groningen, not only did I conquer the challenges of writing this thesis, but I also became a confident biker, cruising along scenic canals, charming shops, and bumping into friendly folks. Speaking of friendly folks, Elina from the Bernoulli Institute deserves a medal for making my administrative tasks and transfer to Groningen as smooth as butter. A big thank you to Fabian for his heroic efforts in making the Dutch summary of this thesis understandable not just to Google Translate, but to actual Dutch speakers as well. Although I didn't have the chance to hang out with the new MINDS crew in Groningen, the few Zoom calls with Steven, Guillaume, and Celestine were filled with fun and inspiration. Of course, I would be remiss if I didn't thank the coffee machines in the Bernoulli Institute for fueling my hot chocolate addiction and keeping me hydrated throughout the writing process.

My genuine gratitude goes to Prof. Gepperth, Prof. Schomaker, Prof. Rish, and Prof. Sabatelli for their approval and constructive feedback on this thesis. I am honored by their valuable time and insights.

Finally, and therefore most importantly and not-so-academically, I want to thank my friends and family for their support and companionship during this wild chapter of my life. Studying abroad can be a daunting experience for many, but not for me. I was lucky to have my host-father, Jens, who made the international metropolis Etelsen a place I could call home. His endless supply of Brötchen and Sinalco fueled my brain to tackle even the most challenging research problems. I will never forget the countless conversations we had over hot pot and the memorable moments we shared, from exploring quaint German towns to playing card games together. It was also a pleasure to have Tianmi as my partner in various crimes; he managed to enlighten me with the art of Zen that I had previously enlightened him with. My heartfelt thanks goes to Sophie for sharing most of my happiness and sadness along this roller coaster. She constantly dragged me out of my research cave and reminded me to have a life. Her contagious sense of humor taught me how to see life as a comedy, even in the face of tragedy. As in any other chapter of my life, I owe the most to my parents. How they can unconditionally love and trust a rebellious son is a question worth another PhD investigation.