Continual Learning with Neural Networks

Sayna Ebrahimi

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-82
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-82.html

May 28, 2020

Continual Learning with Neural Networks

by

Sayna Ebrahimi

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair
Professor Jitendra Malik
Professor Alexei Efros

Spring 2020

The thesis of Sayna Ebrahimi, titled Continual Learning with Neural Networks, is approved:

Chair ______________________ Date 5/24/2020

Jmalik ______________________ Date 5/28/2020

______________________ Date 5/26/2020

University of California, Berkeley

Continual Learning with Neural Networks

Abstract

Continual Learning with Neural Networks

by

Sayna Ebrahimi

Master of Science in Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

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

The work described in this thesis has been dedicated to the investigation of continual learning and solutions to mitigate the forgetting phenomena in two common types of neural networks: Bayesian and non-Bayesian frameworks. We assume a task incremental setting where tasks arrive one at a time with distinct boundaries.

First, we start by building an evolving system where the capacity dynamically increases to accommodate new tasks without compromising scalability. We do so by developing a shared knowledge among tasks while learning features unique to each one. To further ensure preventing forgetting we use small episodic memory containing few samples from old tasks. We show this approach for non-Bayesian neural networks without loss of generality and applicability to Bayesian neural networks.

Second, unique to Bayesian neural networks, as an alternative to dynamically grow the architecture and store the old data, which might not be feasible if not impossible due to confidentiality issues, important parameters in a model can be identified and future changes on them get regularized. We consider a fixed network capacity where each parameter is a distribution rather than a real-valued number. We leverage the uncertainty defined per parameters in Bayesian setting to guide the continual learning process in determining the important parameters; the more certain a parameter is, the less we want it to alter in favor of learning new tasks. We show a simple yet effective regularization technique where the learning rate of parameters is inversely scaled with their uncertainty.

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

To my parents, Maryam and Hossein.

Thank you.

Contents

List of Figures

List of Tables

Acknowledgments

The past few years have brought many people across my path, and all have contributed to my journey and accomplishment in different ways. It is important for me to recount and recognize how so many people played a role in the completion of this thesis which is in fact part of the longer journey of my PhD.

First and foremost, I want to express my sincere gratitude to my advisor, Professor Trevor Darrell for taking me in as his advisee.

Trevor trusted my research ability since we met despite my different background and changed my life forever by accepting me to be part of his research group. He guided my research and challenged me to keep my theoretical mind sharp. He was always interested, invested, and actively involved in my research and was always willing to give comments on, be criticism of, and provide suggestion to improve any work I had done. I learned from him how to think broad, yet not losing focus from the ultimate long-term agenda. His consistent reminder for being confident and remaining strong has made me the researcher I am today. One of the biggest strengths Trevor has instilled in me is to not give up. He has been indeed one of the reasons I have kept fighting my cancer during all these years. I cannot thank him enough for being there for me not just as an advisor, but like a caring family member despite being responsible for leading such a large research group. I could always count on him to help me find a way out of any problem. I am thankful for his continued support and guidance as I embark on my academic journey ahead.

I would also like to thank Professor Jitendra Malik and Professor Alexei Efros who both have been there for me as a teacher, mentor, and a committee member. I was fortunate enough to take many of their courses which I have greatly benefited from in various different research projects.

I cannot thank enough Marcus Rohrbach who has been a tremendous help to me throughout my graduate career. Dr. Rohrbach as my supervisor during the 6-month summer internship at Facebook AI Research has been an amazing mentor and taught me great many things that shaped the foundations of my research. I am also thankful to Mohamed Elhoseiny, Franziska Meier, Roberto Calandra for their hospitality and advices during my internships at Facebook.

I am thankful to my labmates, Samaneh Azadi, Anna Rohrbach, Suzie Petryk, Seth Park, Devin Guillory, Erin Grant, Eric Tzang, Juddy Hoffman, Lisa Hendricks, Yang Gao, Parsa Mahmoudieh, Evan Shelhamer, Xin Wang, and Kate Rakelly. In addition to the people that directly informed, guided, and funded my research, many other people greatly helped me in completing this dissertation. First of all, I would like to thank my husband, Behrooz Shahsavari, for his constant understanding and support of my academic goals; he has been an enormous support for me and words cannot express my gratitude for him always being there for me. Last and foremost, I am thanking my parents, my sister and my brother who have given me unconditional love, support and encouragement throughout my life.

Chapter 1

Introduction

Continual learning, also referred to as lifelong learning, or sequential learning, studies the problem of online learning from a stream of data belonging to various tasks which arrives incrementally. The main goal of a continual learning agent is to continue using the acquired knowledge in learning new tasks while being able to maintain its performance on all the acquired knowledge. For such a system or process to be efficient, the previously seen data should not be stored as a whole and a full re-training at each point is simply infeasible at such a large scale.

Humans can learn novel tasks by augmenting core capabilities with new skills learned based on information for a specific novel task. We conjecture that they can leverage a

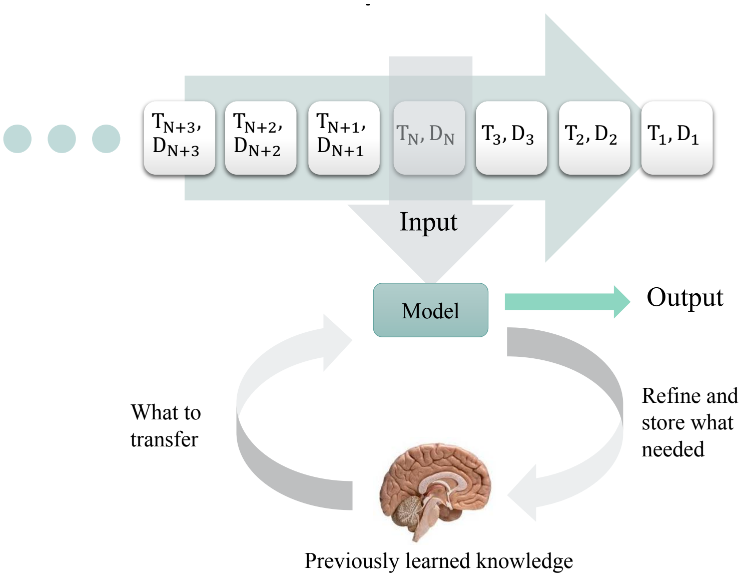


Figure 1.1: An illustration of the continual machine learning cycle. Data is streamed sequentially from different distributions and the continual learning agent tries to transfer knowledge between tasks while retaining its performance on the seen data.

lifetime of previous task experiences in the form of fundamental skills that are robust to different task contexts. When a new task is encountered, these generic strategies form a base set of skills upon which task-specific learning can occur. We would like artificial learning agents to have the ability to solve many tasks sequentially under different conditions by developing task-specific and task-invariant skills that enable them to quickly adapt while avoiding *catastrophic forgetting* (McCloskey & Cohen, 1989a) using their memory.

Since the early development of neural networks, researchers studied the catastrophic forgetting problem and proposed that the parameters sharing which allows neural networks to generalize from seen data is the reason behind catastrophic forgetting.

1.1 Continual Learning Approaches

The existing approaches to prevent catastrophic forgetting can be broadly divided into three categories: memory-based methods, structure-based methods, and regularization-based methods.

Memory-based methods: Methods in this category mitigate forgetting by relying on storing previous experience explicitly or implicitly wherein the former raw samples ((Chaudhry *et al.,* 2019a; Lopez-Paz *et al.,* 2017; Robins, 1995; Rebuffi *et al.,* 2017; Riemer *et al.,* 2018) are saved into the memory for *rehearsal* whereas in the latter a generative model such as a GAN (Shin *et al.,* 2017) or an autoencoder (Kemker & Kanan, 2018) synthesizes them to perform *pseudo-rehearsal*. These methods allow for simultaneous multi-task learning on i.i.d. data which can significantly reduce forgetting. A recent study on tiny episodic memories in CL (Chaudhry *et al.,* 2019b) compared methods such as GEM (Lopez-Paz *et al.,* 2017), A-GEM (Chaudhry *et al.,* 2019a) , MER (Riemer *et al.,* 2018), and ER-RES (Chaudhry *et al.,* 2019b). Similar to (Riemer *et al.,* 2018), for ER-RES they used reservoir sampling using a single pass through the data. Reservoir sampling (Vitter, 1985) is a better sampling strategy for long input data compared to random selection. In this work, we explicitly store raw samples into a very tiny memory used for replay buffer and we differ from prior work in this line of research by how these stored examples are used by specific parts of our model (discriminator and the shared module) to prevent forgetting in the features found to be shared across tasks.

Structure-based methods: These methods exploit modularity and attempt to localize inference to a subset of the network such as modules (Rusu *et al.,* 2016; Ebrahimi *et al.,* 2020b), neurons (Fernando *et al.,* 2017; Yoon *et al.,* 2018), a mask over parameters (Mallya & Lazebnik, 2018; Serra *et al.,* 2018). The performance on previous tasks is preserved by storing the learned module while accommodating new tasks by augmenting the network with new modules. For instance, Progressive Neural Nets (PNNs) (Rusu *et al.,* 2016) statically grow the architecture while retaining lateral connections to previously frozen modules resulting in guaranteed zero forgetting at the price of quadratic scale in the number of parameters. (Yoon

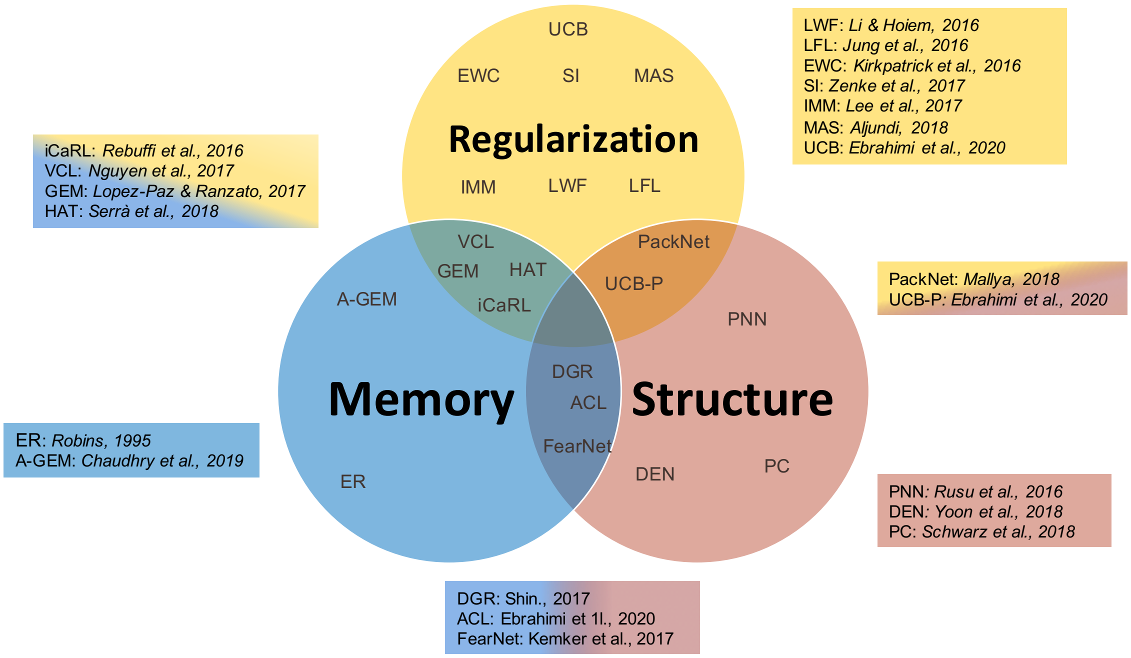


Figure 1.2: A Venn diagram illustrating the different continual learning approaches and their overlaps.

et al., 2018) proposed dynamically expandable networks (DEN) in which, network capacity grows according to tasks *relatedness* by splitting/duplicating the most important neurons while time-stamping them so that they remain accessible and re-trainable at all time. This strategy despite introducing computational cost is inevitable in continual learning scenarios where a large number of tasks are to be learned and a fixed capacity cannot be assumed.

Regularization methods: In these methods (Kirkpatrick *et al.*, 2017; Zenke *et al.*, 2017; Aljundi *et al.*, 2018; Ebrahimi *et al.*, 2020a), the learning capacity is assumed fixed and continual learning is performed such that the change in parameters is controlled and reduced or prevented if it causes performance downgrade on prior tasks. Therefore, for parameter selection, there has to be defined a *weight importance* measurement concept to prioritize parameter usage. For instance, inspired by Bayesian learning, in elastic weight consolidation (EWC) method (Kirkpatrick *et al.*, 2017) important parameters are those to have the highest in terms of the Fisher information matrix. HAT (Serra *et al.*, 2018) learns an attention mask over *important* parameters. In (Ebrahimi *et al.*, 2020a) we used per-weight uncertainty defined in Bayesian neural networks to control the change in parameters. Despite the success gained by these methods in maximizing the usage of a fixed capacity, they are often limited by the number of tasks. In this work we propose two general continual learning frameworks for Bayesian and non-Bayesian neural networks:

In Chapter 2, we hypothesize that representations learned to solve each task in a sequence have a shared structure while containing some task-specific properties. We show that shared features are significantly less prone to forgetting and propose a novel hybrid continual learning framework that learns a disjoint representation for task-invariant and task-specific features required to solve a sequence of tasks. Our model combines architecture growth to prevent forgetting of task-specific skills and an experience replay approach to preserve shared skills. We demonstrate our hybrid approach is effective in avoiding forgetting and show it is superior to both architecture-based and memory-based approaches on class incrementally learning of a single dataset as well as a sequence of multiple datasets in image classification.

In Chapter 3, we propose a simple yet effective regularization-based approach withing Bayesian neural networks. Current regularization-based continual learning algorithms need an external representation and extra computation to measure the parameters' importance. In contrast, we propose Uncertainty-guided Continual Bayesian Neural Networks (UCB), where the learning rate adapts according to the uncertainty defined in the probability distribution of the weights in networks. Uncertainty is a natural way to identify what to remember and what to change as we continually learn, allowing to mitigate catastrophic forgetting. We also show a variant of our model, which uses uncertainty for weight pruning and retains task performance after pruning by saving binary masks per tasks. We evaluate our UCB approach extensively on diverse object classification datasets with short and long sequences of tasks and report superior or on-par performance compared to existing approaches. Additionally, we show that our model does not necessarily need task information at test time, i.e., it does not presume knowledge of which task a sample belongs to.

Chapter 2

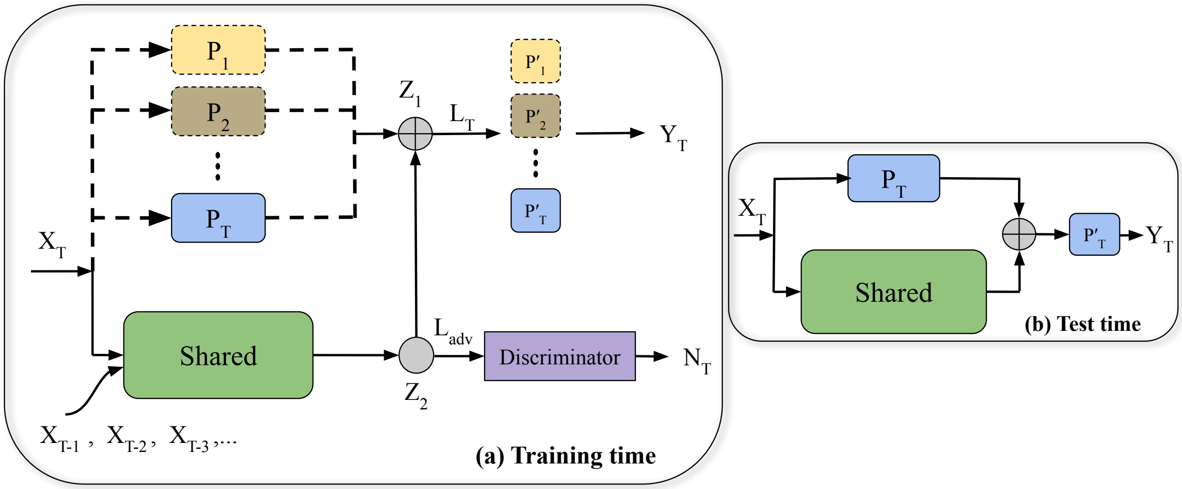
Continual Learning in Ordinary Neural Networks

2.1 Introduction

One line of continual learning approaches learns a single representation with a fixed capacity in which they detect important weight parameters for each task and minimize their further alteration in favor of learning new tasks. In contrast, structure-based approaches increase the capacity of the network to accommodate new tasks. However, these approaches do not scale well to the large number of tasks if they require a large amount of memory for each task. Another stream of approaches in continual learning relies on explicit or implicit experience replay by storing raw samples or training generative models, respectively.

In this chapter we propose a novel adversarial continual learning (ACL) method in which a disjoint latent space representation composed of *task-specific* or *private* latent space is learned for each task and a *task-invariant* or *shared* feature space is learned for all tasks to enhance better knowledge transfer as well as better recall of the previous tasks. The intuition behind our method is that tasks in a sequence share a part of the feature representation but also have a part of the feature representation which is task-specific. The shared features are notably less prone to forgetting and the tasks-specific features are important to retain to avoid forgetting the corresponding task. Therefore, factorizing these features separates the part of the representation that forgets from that which does not forget. To disentangle the features associated with each task, we propose a novel adversarial learning approach to enforce the shared features to be task-invariant and employ orthogonality constraints (Salzmann *et al.,* 2010) to enforce the shared features to not appear in the task-specific space.

Adversarial learning has been used for different problems such as generative models (Goodfellow *et al.,* 2014), object composition (Azadi *et al.,* 2018; "Compositional GAN: Learning Image-Conditional Binary Composition"), representation learning (Makhzani *et al.,* 2015), domain adaptation (Tzeng *et al.,* 2017), active learning (Sinha *et al.,* 2019; "Variational



(a) Training time

(b) Test time

Figure 2.1: Factorizing task-specific and task-invariant features in our method (ACL). *Left:* Shows ACL at training time where the Shared module is adversarially trained with the discriminator to generate *task-invariant* features ($\mathbf{z}_S$) while the discriminator attempts to predict task labels. Architecture growth occurs at the arrival of each task by adding a *task-specific* module optimized to generate orthogonal representation ($\mathbf{z}_P$) to $\mathbf{z}_S$. To prevent forgetting, 1) Private modules are stored for each task and 2) A shared module which is less prone to forgetting, yet is also retrained with experience reply with a limited number of exemplars *Right:* At test time, discriminator is removed and ACL uses the P module for the specific task it is evaluated on.

Adversarial Active Learning"), etc. The use of an adversarial network enables the model to train in a fully-differential manner by adjusting to solve the *minimax* optimization problem (Goodfellow *et al.*, 2014). Adversarial learning of the latent space has been extensively researched in domain adaptation (Hoffman *et al.*, 2018), and representation learning (Kim & Mnih, 2018; Makhzani *et al.*, 2015). While previous literature is concerned with the case of modeling a single or multiple tasks at once (Ebrahimi *et al.*, 2017a; Ebrahimi *et al.*, 2017b), with few examples (Schonfeld *et al.*, 2019a; Schönfeld *et al.*, 2019; Schonfeld *et al.*, 2019b) here we extend this literature by considering the case of continuous learning where multiple tasks need to be learned in a sequential manner.

Once factorization is complete, minimizing forgetting in each space can be handled differently. In the task-specific latent space, due to the importance of these features in recalling the task, we freeze the private module and add a new one upon finishing learning a task. The shared module, however, is significantly less susceptible to forgetting and we only use the replay buffer mechanism in this space to the extend that factorization is not perfect, i.e., when tasks have little overlap or have high domain shift in between, using a tiny memory containing samples stored from prior tasks will help with better factorization and hence higher performance. We empirically found that unlike other memory-based methods in which performance increases by increasing the samples from prior tasks, our model requires a very tiny memory budget beyond which its performance remains constant. This alleviates the need to use old data, as in some applications it might not be possible to store a large amount of data, if any at all, after observing it. Instead, our approach leaves room for further use of memory, if available and need be, for architecture growth. Our approach is simple yet surprisingly powerful in not forgetting and achieves state-of-the-art results on visual continual learning benchmarks such as MNIST, CIFAR100, Permuted MNIST, and miniImageNet.

2.2 Adversarial Continual learning (ACL)

We consider the problem of learning a sequence of T data distributions denoted as $\mathcal{D}^{tr} = \{\mathcal{D}_1^{tr}, \cdots, \mathcal{D}_T^{tr}\}$, where $\mathcal{D}_k^{tr} = \{(\mathbf{X}_i^k, \mathbf{Y}_i^k, \mathbf{T}_i^k)_{i=1}^{n_k}\}$ is the data distribution for task k with n sample tuples of input ($\mathbf{X}^k \in \mathcal{X}$), output label ($\mathbf{Y}^k \in \mathcal{Y}$), and task label ($\mathbf{T}^k \in \mathcal{T}$). The goal is to sequentially learn the model $f_\theta : \mathcal{X} \to \mathcal{Y}$ for each task that can map each task input to its target output while maintaining its performance on all prior tasks. We aim to achieve this by learning a disjoint latent space representation composed of a *task-specific* latent space for each task and a *task-invariant* feature space for all tasks to enhance better knowledge transfer as well as better catastrophic forgetting avoidance of prior knowledge. We mitigate catastrophic forgetting in each space differently. For the *task-invariant* feature space, we assume a limited memory budget of $\mathcal{M}^k$ which stores m samples $x_{i=1\cdots m} \sim \mathcal{D}_{j=1\cdots k-1}^{tr}$ from every single task prior to k.

We begin by learning f_θ^k as a mapping from $\mathbf{X}^k$ to $\mathbf{Y}^k$. For C-way classification task with a cross-entropy loss, this corresponds to

$$\mathcal{L}_{\text{task}}(f_\theta^k, \mathbf{X}^k, \mathbf{Y}^k, \mathcal{M}^k) = -\mathbb{E}_{(x^k, y^k) \sim (\mathbf{X}^k, \mathbf{Y}^k) \cup \mathcal{M}^k} \sum_{c=1}^{C} \mathbb{1}_{[c=y^k]} \log(\sigma(f_\theta^k(x^k))) \tag{2.1}$$

where σ is the softmax function and the subscript $i = \{1, \cdots, n_t\}$ is dropped for simplicity. In the process of learning a sequence of tasks, an ideal f^k is a model that maps the inputs to two independent latent spaces where one contains the shared features among all tasks and the other remains private to each task. In particular, we would like to disentangle the latent space into the information shared across all tasks ($\mathbf{z}_S$) and the independent or private information of each task ($\mathbf{z}_P$) which are as distinct as possible while their concatenation followed by a task-specific head outputs the desired targets.

To this end, we introduce a mapping called Shared ($S_{\theta_S} : \mathcal{X} \to \mathbf{z}_S$) and train it to generate features that fool an adversarial discriminator D. Conversely, the adversarial discriminator ($D_{\theta_D} : \mathbf{z}_S \to \mathcal{T}$) attempts to classify the generated features by their task labels ($\mathbf{T}^{k \in \{0, \cdots, T\}}$) . This is achieved when the discriminator is trained to maximize the probability of assigning the correct task label to generated features while simultaneously S is trained to confuse the discriminator by minimizing $\log(D(S(x^k)))$. This corresponds to the following T-way classification cross-entropy adversarial loss for this minimax game

$$\mathcal{L}_{\text{adv}}(D, S, \mathbf{X}^k, \mathbf{T}^k, \mathcal{M}^k) = \min_S \max_D \sum_{k=0}^{T} \mathbb{1}_{[k=t^k]} \log \left(D \left(S \left(x^k \right) \right) \right) \ . \tag{2.2}$$

Note that the extra label zero is associated with the 'fake' task label paired with randomly generated noise features of $\mathbf{z}'_S \sim \mathcal{N}(\mu, \sum)$. In particular, we use adversarial learning in a different regime that appears in most works related to generative adversarial networks (Goodfellow *et al.,* 2014) such that the generative modeling of input data distributions is not utilized here because the ultimate task is to learn a discriminative representation.

2.2.1 Orthogonality Constraint

In the machine learning literature, *multi-view* learning, aims at constructing and/or using different views or modalities for better learning performances (Blum & Mitchell, 1998; Xu *et al.,* 2013). The approaches to tackle multi-view learning aim at either maximizing the mutual agreement on distinct views of the data, or focus on obtaining a latent subspace shared by multiple views by assuming that the input views are generated from this latent subspace using Canonical correlation analysis and clustering (Chaudhuri *et al.,* 2009), Gaussian processes (Shon *et al.,* 2006), etc. Therefore, the concept of factorizing the latent space into *shared* and *private* parts has been extensively explored for different data modalities. Inspired by the practicality of factorized representation in handling different modalities, here we factorize the latent space learned for different tasks using adversarial learning and orthogonality constraints (Salzmann *et al.,* 2010).

To facilitate training S, we use the Gradient Reversal layer (Ganin *et al.,* 2016) that optimizes the mapping to maximize the discriminator loss directly ($\mathcal{L}_{\text{task}_S} = -\mathcal{L}_{\text{D}}$). In fact, it acts as an identity function during forward propagation but negates its inputs and reverses the gradients during back propagation. The training for S and D is complete when S is able to generate features that D can no longer predict the correct task label for leading $\mathbf{z}_S$ to become as task-invariant as possible. The private module ($P_{\theta_P} : \mathcal{X} \to \mathbf{z}_P$), however, attempts to accommodate the task-invariant features by learning merely the features that are specific to the task in hand and do not exist in $\mathbf{z}_S$. We further factorize $\mathbf{z}_S$ and $\mathbf{z}_P$ by using orthogonality constraints introduced in (Salzmann *et al.,* 2010), also known as "difference" loss in the domain adaptation literature (Bousmalis *et al.,* 2016), to prevent the shared features between all tasks from appearing in the private encoded features. This corresponds to

$$\mathcal{L}_{\text{diff}}(S, P, \mathbf{X}^k, \mathcal{M}^k) = \sum_{k=1}^{T} ||(S(x^k))^{\mathrm{T}} P^k(x^k)||_F^2 \,, \tag{2.3}$$

where $|| \cdot ||_F$ is the Frobenius norm and it is summed over the encoded features of all P modules encoding samples for the current tasks and the memory.

Final output predictions for each task are then predicted using a task-specific multi-layer perceptron head which takes $\mathbf{z}_P$ concatenated with $\mathbf{z}_S$ $((\mathbf{z}_P \oplus \mathbf{z}_S))$ as an input.

Taken together, these loss form the complete objective for ACL as

$$\mathcal{L}_{\text{ACL}} = \lambda_1 \mathcal{L}_{\text{adv}} + \lambda_2 \mathcal{L}_{\text{task}} + \lambda_3 \mathcal{L}_{\text{diff}}\,, \tag{2.4}$$

where λ_1, λ_2, and λ_3 are regularizers to control the effect of each component. The full algorithm for ACL is given in Alg. 1.

2.2.2 Avoiding forgetting in ACL

Catastrophic forgetting occurs when a representation learned through a sequence of tasks changes in favor of learning the current task resulting in performance downgrade on previous tasks. The main insight to our approach is decoupling the conventional *single* representation learned for a sequence of tasks into two parts: a part that *must not change* because it contains task-specific features without which complete performance retrieval is not possible, and a part that is *less prone to change* as it contains the core structure of all tasks.

To *fully prevent* catastrophic forgetting in the first part (private features), we use *compact* modules that can be stored into memory. If factorization is successfully performed, the second part remains highly immune to forgetting. However, we empirically found that when disentanglement cannot be fully accomplished either because of the little overlap or large domain shift between the tasks, using a tiny replay buffer containing few samples for old data can be beneficial to retain high ACC values as well as mitigating forgetting.

2.2.3 Evaluation metrics

After training for each new task, we evaluate the resulting model on all prior tasks. Similar to (Lopez-Paz *et al.*, 2017; Ebrahimi *et al.*, 2020a), to measure ACL performance we use ACC as the average test classification accuracy across all tasks. To measure forgetting we report backward transfer, BWT, which indicates how much learning new tasks has influenced the performance on previous tasks. While BWT < 0 directly reports *catastrophic forgetting*, BWT > 0 indicates that learning new tasks has helped with the preceding tasks.

$$\text{BWT} = \frac{1}{T-1} \sum_{i=1}^{T-1} R_{T,i} - R_{i,i}, \quad \text{ACC} = \frac{1}{T} \sum_{i=1}^{T} R_{T,i} \tag{2.5}$$

where $R_{n,i}$ is the test classification accuracy on task i after sequentially finishing learning the n^{th} task.

We also compare methods based on the memory used either in the network architecture growth or replay buffer. Therefore, we convert them into memory size assuming numbers are 32-bit floating point which is equivalent to 4bytes.

Algorithm 1 Adversarial Continual Learning (ACL)

```
procedure TRAIN$(()\theta_P, \theta_S, \theta_D, \mathcal{D}^{tr}, \mathcal{D}^{ts}, m)$
    Hyper-parameters: $\lambda_1, \lambda_2, \lambda_3, \alpha_S, \alpha_P, \alpha_D$
    $R \leftarrow 0 \in \mathbb{R}^{T \times T}$
    $\mathcal{M} \leftarrow \{\}$
    $f_\theta^k = f(\theta_S \oplus \theta_P)$
    for $k = 1$ to T do
        for $e = 1$ to epochs do
            Compute $\mathcal{L}_{\text{adv}}$ for $S$ using $(x, t) \in \mathcal{D}_k^{tr} \cup \mathcal{M}$
            Compute $\mathcal{L}_{\text{task}}$ using $(x, y) \in \mathcal{D}_k^{tr} \cup \mathcal{M}$
            Compute $\mathcal{L}_{\text{diff}}$ using $P^k$, $S$, and $x \in \mathcal{D}_k^{tr}$
            $\mathcal{L}_{\text{ACL}} = \lambda_1 \mathcal{L}_{\text{adv}} + \lambda_2 \mathcal{L}_{\text{task}} + \lambda_3 \mathcal{L}_{\text{diff}}$
            $\theta'_S \leftarrow \theta_S - \alpha_S \nabla \mathcal{L}_{\text{ACL}}$
            $\theta'_{P_t} \leftarrow \theta_{P^k} - \alpha_{P^k} \nabla \mathcal{L}_{\text{ACL}}$
            Compute $\mathcal{L}_{\text{adv}}$ for $D$ using $(S(x), t)$ and $(\mathbf{z}' \sim \mathcal{N}(\mu, \sum), t = 0)$
            $\theta'_{\text{D}} \leftarrow \theta_D - \alpha_D \nabla \mathcal{L}_{\text{adv}}$
        $\mathcal{M} \leftarrow$ UPDATEMEMORY$(\mathcal{D}_k^{tr}, \mathcal{M}, C, m)$
        Store $\theta_{P^k}$
        $f_\theta^k \leftarrow f(\theta'_S \oplus \theta'_P)$
        $R_{k,\{1\cdots k\}} \leftarrow$ EVAL $(f_\theta^k, \mathcal{D}^{ts}_{\{1\cdots k\}})$
```

```
procedure UPDATEMEMORY$(()\mathcal{D}_k^{tr}, \mathcal{M}, C, m)$
    $s \leftarrow \frac{m}{C}$                    ▷ $s :=$ # of samples per class
    for $c = 1$ to $C$ do
        for $i = 1$ to $n$ do
            $(x_i^k, y_i^k, t_i^k) \sim \mathcal{D}_k^{tr}$
            $\mathcal{M} \leftarrow \mathcal{M} \cup (x^k, y^k, t^k)$
    return $\mathcal{M}$

procedure EVAL$(()f_\theta^k, \mathcal{D}^{ts}_{\{1\cdots k\}})$
    for $i = 1$ to $k$ do
        $R_{k,i} =$ Accuracy$(f_\theta^k(x, t), y)$for$(x, y, t) \in \mathcal{D}_i^{ts}$
    return $R$
```

2.3 Experiments

In this section, we review the benchmark datasets and baselines used in our evaluation as well as the implementation details. We then report the obtained results, an ablation study, and a brief analysis of ACL details.

2.3.1 ACL on Vision Benchmarks

Datasets: We evaluate our approach on the commonly used benchmarks datasets for T-split class-incrementally learning where the entire dataset is divided into T disjoint susbsets or tasks.

We use common image classification datasets **5-Split MNIST** and **Permuted MNIST** (LeCun *et al.*, 1998), previously used in (Nguyen *et al.*, 2018; Zenke *et al.*, 2017; Ebrahimi *et al.*, 2020a), **20-Split CIFAR100** (Krizhevsky & Hinton, 2009) used in (Zenke *et al.*, 2017; Lopez-Paz *et al.*, 2017; Chaudhry *et al.*, 2019a), and **20-Split miniImageNet** (Vinyals *et al.*, 2016) used in (Chaudhry *et al.*, 2019b; Zhang *et al.*, 2019). We also benchmark ACL on a sequence of **5-Datasets** including **SVHN**, **CIFAR10**, **not-MNIST**, **Fashion-MNIST** and, **MNIST** and report average performance over multiple random task orderings. Dataset statistics are given in Table 2.3b in the appendix. No data augmentation of any kind has been used in our analysis.

Baselines: From the prior work, we compare with state-of-the-art including Elastic Weight Consolidation (EWC) (Kirkpatrick *et al.*, 2017), Progressive neural networks (PNNs) (Rusu *et al.*, 2016), and Hard Attention Mask (HAT) (Serra *et al.*, 2018) using implementations provided by (Serra *et al.*, 2018) unless otherwise stated. For memory-based methods including A-GEM, GEM, and ER-RES, for Permuted MNIST, 20-Split CIFAR100, and 20-Split miniImageNet, we relied on the implementation provided by (Chaudhry *et al.*, 2019b), but changed the experimental setting from single to multi-epoch and without using 3 Tasks for cross validation for a more fair comparison against ACL and other baselines. On Permuted MNIST results for SI (Zenke *et al.*, 2017) are reported from (Serra *et al.*, 2018), for VCL (Nguyen *et al.*, 2018) those are obtained using their original provided code, and for Uncertainty-based CL in Bayesian framework (UCB) (Ebrahimi *et al.*, 2020a) are directly reported from the paper.

We also perform fine-tuning, and joint training. In fine-tuning (ORD-FT), an ordinary single module network without the discriminator is continuously trained without any forgetting avoidance strategy in the form of experience replay or architecture growth. In joint training with an ordinary network (ORD-JT) and our ACL setup (ACL-JT) we learn all the tasks jointly in a multitask learning fashion using the entire dataset at once which serves as the upper bound for average accuracy on all tasks, as it does not adhere to the continual learning scenario.

Implementation details: For all ACL experiments except for Permuted MNIST and 5-Split MNIST we used a reduced AlexNet (Iandola *et al.*, 2016) architecture as the backbone for S and P modules. The architecture in S is composed of 3 convolutional and 4 fully-connected (FC) layers whereas P is only a convolutional neural network (CNN) with similar number of layers and half-sized kernels compared to those used in S. The private head modules (p) and the discriminator are all composed of a small 3-layer perceptron. Due to the differences between the structure of our setup and a regular network with a single module, we used a similar CNN structure to S followed by larger hidden FC layers to match the total number of parameters throughout our experiments with our baselines for fair comparisons. For 5-Split MNIST and Permuted MNIST where baselines use a two-layer perceptron with 256 units in each and ReLU nonlinearity, we used a two-layer perceptron of size 784×175 and 175×128 with ReLU activation in between in the shared module and a single-layer of size 784×128 and ReLU for each P. In each head, we also used an MLP with layers of size 256 and 28, ReLU activations, and a 14-unit softmax layer. In all our experiments, no pre-trained model is used. We used stochastic gradient descent for ACL and baselines. Our code is provided as a zipped file included in the supplementary materials.

2.4 Results and Discussion

In the first set of experiments, we measure ACC, BWT, and the memory used by ACL and compare it against state-of-the-art methods with or without memory constraints on

20-Split miniImageNet. Next, we provide more insight and discussion on ACL and its component by performing an ablation study and visualizations on this dataset. In Section 2.5, we evaluate ACL on a more difficult continual learning setting where we sequentially train on 5 different datasets. Finally, in section (2.6), we demonstrate the experiments on class incremental learning of single datasets commonly used in CL literature and compare the ACC and BWT metrics against prior work.

2.4.1 ACL Performance on 20-Split miniImageNet

Starting with 20-Split miniImageNet, we split it in 20 tasks with 5 classes at a time. Table 2.1a shows our results obtained for ACL compared to several baselines. We compare ACL with HAT as a regularization based method with no experience replay memory dependency that achieves ACC=59.45 ± 0.05 with BWT=-0.04 ± 0.03%. Results for the memory-based methods of ER-RES and A-GEM are re(produced) by us using the implementation provided in (Chaudhry *et al.*, 2019b) by applying modifications to the network architecture to match with ACL in the backbone structure as well as the number of parameters. We only include A-GEM in Table 2.1a which is only a faster algorithm compared to its precedent GEM with identical performance.

A-GEM and ER-RES use an architecture with 25.6 parameters (102.6MB) along with storing 13 images of size $(84 \times 84 \times 3)$ per class (110.1MB) resulting in total memory size of 212.7MB. ACL is able to outperform all baselines in ACC=$\mathbf{62.07 \pm 0.51}$, BWT=$\mathbf{0.00 \pm 0.00}$, using total memory of 121.6MB for architecture growth (113.1MB) and storing 1 sample per class for replay buffer (8.5MB). In our ablation study in Section 2.4.2, we will show our performance without using replay buffer for this dataset is ACC=57.66 ± 1.44. However, ACL is able to overcome the gap by using only one image per class (5 per task) to achieve ACC=$\mathbf{62.07 \pm 0.51}$ without the need to have a large buffer for old data in class incrementally learning datasets like miniImagenet with diverse sets of classes.

2.4.2 Ablation Studies on 20-Split miniImageNet

We now analyze the major building blocks of our proposed framework such as the discriminator, the shared module, replay buffer effect and the *difference* loss on the miniImagenet dataset. Ablation results are summarized in Table 2.1b and are as follows:

Discriminator and shared modules: We begin by ablating the discriminator and shared module (w/o Dis and Shared) which means only using the private modules. They are used one at a time for each task and stored in memory for further recall (zero-forgetting guaranteed). In 20-Split miniImageNet experiment, we use a small convolutional network with 943.1K parameters in P and 74.2K in p (private head) for each task. The average accuracy (ACC) for this model is 29.09 ± 5.67% where random chance is 20% as it is a 5-class problem. This ablation shows that despite obtaining zero forgetting using P modules

Table 2.1: CL results on 20-Split miniImageNet measuring ACC (%), BWT (%), and Memory (MB). (**) denotes that methods do not adhere to the continual learning setup: ACL-JT and ORD-JT serve as the upper bound for ACC for ACL/ORD networks, respectively. * denotes result is re(produced) by us using the original provided code. † denotes result is obtained using the re-implementation setup by (Serra *et al.*, 2018). All results are averaged over 3 runs and standard deviation is given in parentheses (b) Ablation study of ACL on miniImageNet dataset

(a)

Method	**ACC%**	**BWT%**	Arch (MB)	Replay Buffer (MB)
HAT*(Serra *et al.*, 2018)	59.45(0.05)	-0.04(0.03)	123.6	-
PNN † (Rusu *et al.*, 2016)	58.96(3.50)	Zero	588	-
ER-RES*	57.32(2.56)	-11.34(2.32)	102.6	110.1
A-GEM* (Chaudhry *et al.*, 2019a)	52.43(3.10)	-15.23(1.45)	102.6	110.1
ORD-FT**	28.76(4.56)	-64.23(3.32)	37.6	-
ORD-JT**	69.56(0.78)	-	5100	-
ACL-JT**	66.89(0.32)	-	5100	-
ACL (Ours)	**62.07(0.51)**	**0.00(0.00)**	113.1	8.5

(b)

	ACC%	**BWT%**
ACL (1 sample)	**62.07(0.51)**	**0.00(0.00)**
w/o Dis and Shared	29.09(5.67)	Zero
w/o Private	32.82(2.71)	-28.67(3.61)
w/o $\mathcal{L}_{\text{adv}}$ (w/o Dis)	52.07(2.49)	-0.01(0.01)
w/o Replay buffer	57.66(1.44)	-3.71(1.31)
w/o $\mathcal{L}_{\text{diff}}$	60.28(0.52)	0.00(0.00)

one at a time, they are not capable of performing the task when solely used because of their limited capacity.

Private modules: In the first ablation we showed that none of the tasks can be performed well using only P modules. Ablating the Private modules leads to a similar observation for the shared module as it has only 943.1K parameters. We have performed this ablation by fine-tuning the S module while it is adversarially trained with D. This results in a ACC= 32.82% and BWT=-28.67% confirming the important role of private modules in

Table 2.2: Comparison of the effect of the replay buffer size between ACL and other baselines including A-GEM (Chaudhry *et al.,* 2019a), and ER-RES (Chaudhry *et al.,* 2019b) on 20-Split miniImageNet where unlike the baselines, ACL's performance remains unaffected by the increase in number of samples stored per class as discussed in 2.4.2. The results from this table are used to generate Fig. 2.2 below.

Samples per class		**1**	**3**	**5**	**13**
A-GEM(Chaudhry *et al.,* 2019a)		45.14(3.42)	49.12(4.69)	50.24(4.56)	52.43(3.10)
ER-RES(Chaudhry *et al.,* 2019b)		40.21(2.68)	46.87(4.51)	53.45(3.45)	57.32(2.56)
ACL (ours)	ACC	62.07(0.51)	61.80(0.50)	61.69(0.61)	61.33(0.40)
	BWT	0.00(0.00)	0.01(0.00)	0.01(0.00)	-0.01(0.02)

retaining tasks' performance.

Discriminator: Now, we ablate the adversarial learning aspect of our method. Eliminating the role of the discriminator and hence the adversarial learning aspect of ACL, results in a 14% drop in ACC. This result demonstrates the important role of D in ACL.

Replay buffer: We then explore the effect of using previous samples as replay buffer in avoiding forgetting. We keep ACL in its full shape (S+P+D) but we do not use samples from previous tasks during training. Note that the discriminator has to still predict task labels for new task data as they become available while it has no chance of seeing the previous examples. By comparing the ACC achieved with no memory access with the best results obtained for ACL with memory access (ACL-1 samples) shows the effect of adding a single image per class (5 per task) in performance gain from 57.66% to 62.07%. Unlike A-GEM, and EP-RES approaches in which performance increases with more episodic memory, in ACL, ACC remains nearly similar to its highest performance. We have visualized this effect in Fig. 2.2 where on the left it illustrates the memory effect for ACL and memory-dependent baselines when 1, 3, 5, and 13 images per class are used during training. Numbers used to plot this figure with their standard deviation are given in Table 2.2. We also show how memory affects the BWT in ACL in Fig. 2.2 (right) which follows the same pattern as we observed for ACC. Being insensitive to the amount of old data is a remarkable feature of ACL, not because of the small memory it consumes, but mainly due to the fact that access to the old data might be prohibited or very limited in some real world applications. Therefore, for a fixed allowed memory size, a method that can effectively use it for architecture growth can be considered as more practical for such applications.

Orthogonality Constraint ($\mathcal{L}_{\text{diff}}$): Finally, we study the effect of orthogonality constraint in our objective function defined in Eq. 2.3. The results show an increase of 2% in ACC compared to when adversarial learning is the only method we use for factorization. Comparing the performance increase resulting by adding $\mathcal{L}_{\text{adv}}$ versus $\mathcal{L}_{\text{diff}}$ confirms that the adversarial learning plays an important role in latent space disentanglement in our approach. We hy-

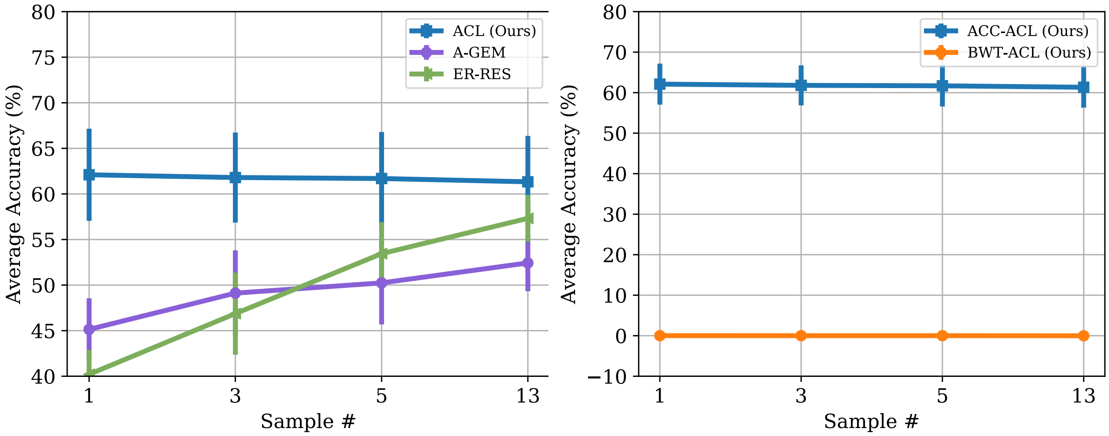


Figure 2.2: *Left:* Comparing the replay buffer effect on ACC on 20-Split miniImageNet achieved by ACL against A-GEM (Chaudhry *et al.*, 2019a) and ER-RES (Chaudhry *et al.*, 2019b) when using 1, 3, 5, and 13 samples per classes within each task discussed in 2.4.2. *Right:* Insensitivity of ACC and BWT to replay buffer in ACL. Best viewed in color.

pothesize this is due to the discriminator having direct access to task labels while training S, whereas *difference* loss performs the minimization using $\mathbf{z}_S$ and $\mathbf{z}_P$ only.

2.4.3 Visualizing the effect of adversarial learning in ACL

Here we illustrate the role of adversarial learning in factorizing the latent space learned for a sequence of tasks in a continual learning setting using the T-distributed Stochastic Neighbor Embedding (T-SNE) (Van Der Maaten, 2014) plots for the 20-Split miniImageNet experiment.

Fig. 2.3 visualizes the latent spaces of the shared and private modules trained with and without the discriminator. In particular, we used the model trained on the entire sequence of 20-Split miniImageNet and evaluated it on the test-sets belonging to tasks #18, #19, and #20 each including 100 images for 5 classes, total of 500 samples, which are color-coded with their class labels.

We first compare the discriminator's effect on the latent space generated by the shared modules. As shown in Fig. 2.3, the shared modules trained with adversarial loss (second column from left), consistently appear as a uniformly mixed distribution of encoded samples belonging to all classes for each task. In contrast, in the generated features by shared modules that were trained without a discriminator (fourth column from left), we observe a non-uniformly distributed mixture of features where small clusters can be found for some classes (*e.g.* tasks #18, #20) showing an entangled representation within each task.

We now move on to show the effect of the discriminator D on the private modules' latent spaces. As can be observed in the third column from left, private modules that were trained

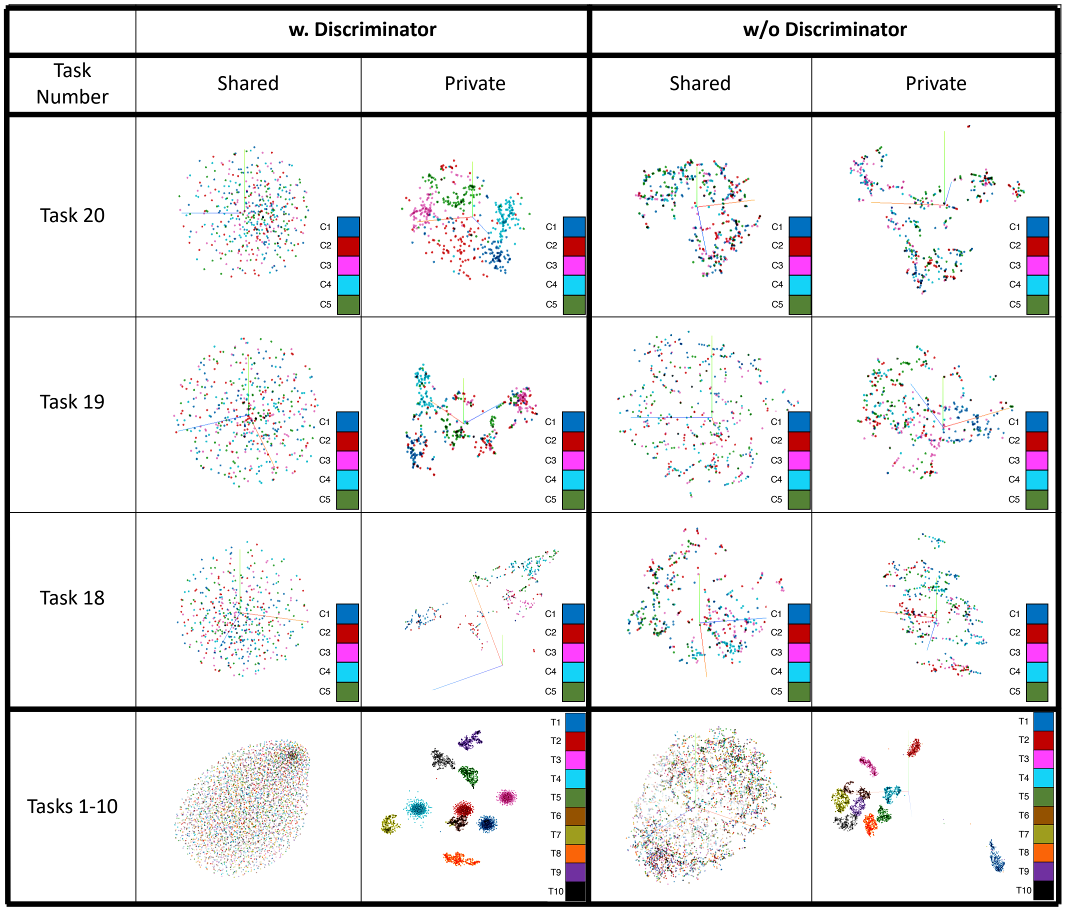


Figure 2.3: Visualizing the effect of adversarial learning in ACL where the latent spaces of both private and shared modules are compared against the generated features by corresponding modules trained without a discriminator. This plot shows that the shared module has been successfully trained to generate task-invariant features using adversarial learning whereas in the fourth column from left, we observe that without the discriminator, the shared module was only able to generate a non-uniform embedding

in a latent space factorized with a discriminator, appear to be nearly successful in uncovering class labels in their latent space although the final classification is yet to be happening in the private heads. As opposed to that, in the absence of the discriminator, private modules (shown in the fifth column) generate features as entangled as those generated by their shared module counterparts.

In the last row of Fig. 2.3, once again we used our final model trained on the entire sequence of 20-Split miniImageNet and tested it on the first 10 tasks of the sequence one at a time and plotted them all in a single figure for both shared and private modules with and without the discriminator. Similar to the pattern we observed above in comparing the shared feature space for each task with/without D, in the first column we observe a uniformly distributed embedding, now color-coded with task labels, where distinguishing tasks is impossible, as expected. In other words, it shows that the shared module has been successfully trained to generate task-invariant features using adversarial learning whereas in the fourth column from the left, we observe that without the discriminator, the shared module was only able to generate a non-uniform embedding. This indicates the impact of the discriminator in finding the true shared features across tasks. On the other hand, for the private module in a setup with a discriminator (third column from left), we observe separate clusters are generated with samples belonging to the same task which shows P is only able to uncover task-specific features when it is trained along with a task-invariant space. Unlike that, a private module that was trained along with a non-adversarial shared module (last column from left), provides nearly similar feature spaces as their shared module counterpart proving that task factorization could not occur without the presence of a discriminator in the setting.

Note that in all the results shown in Fig. 2.3, we did not use the orthogonality constraints, ($\mathcal{L}_{\text{diff}}$), to merely present the role of adversarial learning as the main mechanism used to generate task-specific and task-invariant features.

2.5 ACL Performance on a sequence of 5-Datasets

In this section, we present our results for continual learning of 5 tasks using ACL in Table 2.3b. Similar to the previous experiment we look at both ACC and BWT obtained for ACL, finetuning as well as UCB as our baseline. Results for this sequence are averaged over 5 random permutations of tasks and standard deviations are given in parenthesis. CL on a sequence of datasets has been previously performed by two regularization based approaches of UCB and HAT where UCB was shown to be superior (Ebrahimi *et al.,* 2020a). With this given sequence, ACL is able to outperform UCB by reaching ACC=78.55(±0.29) and BWT$= -0.01$ using only half of the memory size and also no replay buffer. In Bayesian neural networks such as UCB, there exists double number of parameters compared to a regular model representing mean and variance of network weights. It is very encouraging to see that ACL is not only able to continually learn on a single dataset, but also across diverse datasets.

2.6 Additional Experiments

20-Split CIFAR100: In this experiment we incrementally learn CIFAR100 in 5 classes at a time in 20 tasks. As shown in Table 2.3, HAT is the most competitive baseline, although it does not depend on memory and uses 27.2MB to store its architecture in which it learns task-based attention maps reaching ACC$=76.96 \pm 1.23\%$. PNN uses 74.7MB to store the lateral modules to the memory and guarantees zero forgetting. Results for A-GEM, and ER-Reservoir are re(produced) by us using a CNN similar to our shared module architecture. We use fully connected layers with more number of neurons to compensate for the

Table 2.3: CL results on 20-Split CIFAR100 measuring ACC (%), BWT (%), and Memory (MB). (*) denotes that methods do not adhere to the continual learning setup: ACL-JT and ORD-JT serve as the upper bound for ACC for ACL/ORD networks, respectively. † denotes result reported from original work. ‡ denotes result reported from (Lopez-Paz *et al.*, 2017). ** denotes result is reported by (Chaudhry *et al.*, 2019b). †† denotes result is obtained using the re-implementation setup by (Serra *et al.*, 2018). ^{o} denotes result is obtained by using the original provided code. All results are averaged over 3 runs and standard deviation is given in parentheses

(a) 20-Split CIFAR100

Method	**ACC%**	**BWT%**	Arch (MB)	Replay Buffer (MB)
HAT ^{o} (Serra *et al.*, 2018)	76.96(1.23)	0.01(0.02)	27.2	-
PNN†† (Rusu *et al.*, 2016)	75.25(0.04)	Zero	93.51	-
A-GEM** (Chaudhry *et al.*, 2019a)	54.38(3.84)	-21.99(4.05)	25.4	16
ER-RES**	66.78(0.48)	-15.01(1.11)	25.4	16
ORD-FT*	34.71(3.36)	-48.56(3.17)	27.2	-
ORD-JT*	78.67(0.34)	-	764.5	-
ACL-JT*	79.91(0.05)	-	762.6	-
ACL (Ours)	**78.08(1.25)**	**0.00(0.01)**	25.1	-

(b) Sequence of 5 Datasets

Method	**ACC%**	**BWT%**	Arch (MB)	Replay Buffer (MB)
UCB ^{o} (Ebrahimi *et al.*, 2020a)	76.34(0.12)	−1.34(0.04)	32.8	-
ORD-FT*	27.32(2.41)	-42.12(2.57)	16.5	-
ACL (Ours)	**78.55(0.29)**	**−0.01(0.15)**	16.5	-

remaining number of parameters reaching 25.4MB of memory. We also stored 13 images per class (1300 images of size (32 × 32 × 3) in total) which requires 16.0MB of memory. However, ACL achieves ACC=($\mathbf{78.08 \pm 1.25}$)% with BWT=$\mathbf{0.00 \pm 0.01}$)% using only 25.1MB to grow private modules with 167.2K parameters (0.6MB) without using memory for replay buffer. Similar to the previous experiments on MNIST, old data is not used which is mainly due to the overuse of parameters for CIFAR100 which is considered as a relevantly 'easy' dataset with all tasks (classes) sharing the same data distribution. While we leave further discussion about this to Section 2.4.2, we mention that factorizing the shared and private parameters in ACL prevents from using redundant parameters by only storing task-specific

Table 2.4: CL results on Permuted MNIST. measuring ACC (%), BWT (%), and Memory (MB). (*) denotes that methods do not adhere to the continual learning setup: ACL-JT and ORD-JT serve as the upper bound for ACC for ACL/ORD networks, respectively. (†) denotes result reported from original work. (^{o}) denotes result is obtained by using the original provided code. (‡) denotes the results reported by (Serra *et al.*, 2018) and (**) denotes results are reported by (Chaudhry *et al.*, 2019b); T shows the number of tasks. All results are averaged over 3 runs, the standard deviation is provided in parenthesis

Method	**ACC%**	**BWT%**	Arch (MB)	Replay Buffer (MB)
EWC‡ (Kirkpatrick *et al.*, 2017) (T=10)	88.2	-	1.1	-
HAT† (Serra *et al.*, 2018) (T=10)	91.6	-	1.1	-
UCB† (Ebrahimi *et al.*, 2020a) (T=10)	91.44(0.04)	-0.38(0.02)	2.2	-
VCL^o(Nguyen *et al.*, 2018) (T=10)	88.80(0.23)	-7.90(0.23)	1.1	-
VCL-C^o (Nguyen *et al.*, 2018)(T=10)	95.79(0.10)	-1.38(0.12)	1.1	6.3
PNN** (Rusu *et al.*, 2016) (T=20)	93.5(0.07)	Zero	N/A	-
ORD-FT* (T=10)	44.91(6.61)	-53.69(1.91)	1.1	-
ORD-JT* (T=10)	96.03(0.02)	-	189.3	-
ACL-JT* (T=10)	98.45(0.02)	-	194.4	-
ACL (Ours) (T=10)	**98.03(0.01)**	-0.01(0.01)	2.4	-
ACL (Ours) (T=20)	**97.81(0.03)**	0.00(0.00)	5.0	-
ACL (Ours) (T=30)	**97.81(0.03)**	0.00(0.00)	7.2	-
ACL (Ours) (T=40)	**97.80(0.02)**	0.00(0.00)	9.4	-

parameters in P modules. In fact, as opposed to other memory-based methods, instead of starting from a large network and using memory to store samples, which might not be available in practice due to confidentiality issues (*e.g.* medical data), ACL uses memory to gradually add small modules to accommodate new tasks and relies on knowledge transfer through the learned shared module. The latter is what makes ACL to different than architecture-based methods such as PNN where the network grows by the entire *column* which results in using a highly disproportionate memory to what is needed to learn a new task with.

Permuted MNIST: Another popular variant of MNIST dataset in CL literature is Permuted MNIST where each task is composed by randomly permuting pixels of the entire MNIST dataset. To compare against values reported in prior work, we particularly report on a sequence of $T = 10$ and $T = 20$ tasks with ACC, BWT, and memory for ACL and baselines. To further evaluate ACL's ability in handling more tasks, we continually learned up to 40 tasks. As shown in Table 2.4, among the regularization-based methods, HAT achieves the highest performance of 91.6% (Serra *et al.*, 2018) using an architecture of size 1.1MB. Vanilla

Table 2.5: Class Incremental Learning on 5-Split MNIST. measuring ACC (%), BWT (%), and Memory (MB). (*) denotes that methods do not adhere to the continual learning setup: ACL-JT and ORD-JT serve as the upper bound for ACC for ACL/ORD networks, respectively. † denotes result reported from original work. ^{o} denotes result is obtained by using the original provided code. All results are averaged over 3 runs, the standard deviation is provided in parenthesis

Method	**ACC%**	**BWT%**	Arch (MB)	Replay Buffer (MB)
EWC ** (Kirkpatrick *et al.*, 2017)	95.78(0.35)	-4.20(0.21)	1.1	-
HAT ** (Serra *et al.*, 2018)	99.59(0.01)	0.00(0.04)	1.1	-
UCB † (Ebrahimi *et al.*, 2020a)	99.63(0.02)	0.00(0.00)	2.2	-
VCL ^{o}(Nguyen *et al.*, 2018)	95.97(1.03)	-4.62(1.28)	1.1	-
iCaRL‡ (Rebuffi *et al.*, 2017)	89.34(0.40)	-3.24(0.34)	6.5	0.63
GEM^{o} (Lopez-Paz *et al.*, 2017)	94.34(0.82)	-2.01(0.05)	6.5	0.63
VCL-C ^{o} (Nguyen *et al.*, 2018)	93.6(0.20)	-3.10(0.20)	1.7	0.63
ORD-FT*	65.96(3.53)	-40.15(4.27)	1.1	-
ORD-JT*	99.88(0.02)	-	189.3	-
ACL-JT* (Ours)	99.89(0.01)	-	190.8	-
ACL (Ours)	**99.76**(**0.03**)	0.01(0.01)	1.6	-

VCL improves by 7% in ACC and 6.5% in BWT using a K-means core-set memory size of 200 samples per task (6.3MB) and an architecture size similar to HAT. PNN appears as a strong baseline achieving ACC=93.5% with guaranteed zero forgetting. Finetuning (ORD-FT) and joint training (ORD-JT) results for an ordinary network, similar to EWC and HAT (a two-layer MLP with 256 units and ReLU activations), are also reported as reference values for lowest BWT and highest achievable ACC, respectively. ACL achieves the highest accuracy among all baselines for both sequences of 10 and 20 equal to ACC=98.03 ± 0.01 and ACC=97.81 ± 0.03, and BWT= $-$ 0.01% BWT=0%, respectively which shows that performance of ACL drops only by 0.2% as the number of tasks doubles. ACL also remains efficient in using memory to grow the architecture compactly by adding only 55K parameters (0.2MB) for each task resulting in using total of 2.4MB and 5.0MB when $T = 10$ and $T = 20$, respectively for the entire network including the shared module and the discriminator. We also observed that the performance of our model does not change as the number of tasks increases to 30 and 40 if each new task is accommodated with a new private module. Similar to the 5-Split MNIST experiment, we did not store old data and used memory only to grow the architecture by 55K parameters (0.2MB).

5-Split MNIST: As the last experiment in this section, we continually learn $0 - 9$ MNIST digits by following the conventional pattern of learning 2 classes over 5 sequential tasks (Nguyen *et al.*, 2018; Zenke *et al.*, 2017; Ebrahimi *et al.*, 2020a). As shown in Table 2.5, we compare ACL with regularization-based methods with no memory dependency (EWC, HAT, UCB, Vanilla VCL) and methods relying on memory only (GEM and iCaRL), and

VCL with K-means Core-set (VCL-C) where 40 samples are stored per task. ACL reaches ACC=($\mathbf{99.76 \pm 0.03}$)% with zero forgetting outperforming UCB with ACC=99.63% which uses nearly 40% more memory size. In this task, we only use architecture growth (no experience replay) where 54.3K private parameters are added for each task resulting in memory requirement of 1.6MB to store all private modules. Our core architecture has a total number of parameters (420.1K). We also provide naive finetuning results for ACL and a regular single-module network with (268K) parameters (1.1MB). Joint training (multi-task learning) results for the regular network (ORD-JT) is computed as ACC=$99.89\pm, 0.01$ for ACL which requires 189.3MB for the entire dataset as well as the architecture. Joint training only serves as an upper-bound and is not a continual learning baseline.

2.7 Summary

In this work, we proposed a novel hybrid continual learning algorithm – Adversarial Continual Learning (ACL) – that factorizes the representation learned for a sequence of tasks into *task-specific* and *task-invariant* features where the former is important to be fully preserved to avoid forgetting and the latter is empirically found to be remarkably less prone to forgetting. To the best of our knowledge this is the first work that uses adversarial learning along with orthogonality constraints to disentangle the shared and private latent representations which results in compact private modules that can be stored into memory and hence, efficiently preventing forgetting. To further improve mitigating forgetting problem in the shared module, we used a small memory replay buffer. However, this additional memory usage is not critical in our approach.

Chapter 3

Continual Learning in Bayesian Neural Networks

3.1 Introduction

Humans can easily accumulate and maintain knowledge gained from previously observed tasks, and continuously learn to solve new problems or tasks. Artificial learning systems typically forget prior tasks when they cannot access all training data at once but are presented with task data in sequence.

Overcoming these challenges is the focus of *continual learning*, sometimes also referred to as *lifelong learning* or *sequential learning*. *Catastrophic forgetting* (McCloskey & Cohen, 1989b; McClelland *et al.*, 1995) refers to the significant drop in the performance of a learner when switching from a trained task to a new one. This phenomenon occurs because trained parameters on the initial task change in favor of learning new objectives.

Given a network of limited capacity, one way to address this problem is to identify the importance of each parameter and penalize further changes to those parameters that were deemed to be important for the previous tasks (Kirkpatrick *et al.*, 2017; Aljundi *et al.*, 2018; Zenke *et al.*, 2017). An alternative is to freeze the most important parameters and allow future tasks to only adapt the remaining parameters to new tasks (Mallya & Lazebnik, 2018). Such models rely on the explicit parametrization of importance. We propose here implicit uncertainty-guided importance representation.

Bayesian approaches to neural networks (MacKay, 1992b) can potentially avoid some of the pitfalls of explicit parameterization of importance in regular neural networks. Bayesian techniques, naturally account for uncertainty in parameters estimates. These networks represent each parameter with a distribution defined by a mean and variance over possible values drawn from a shared latent probability distribution (Blundell *et al.*, 2015). Variational inference can approximate posterior distributions using Monte Carlo sampling for gradient estimation. These networks act like ensemble methods in that they reduce the prediction variance but only use twice the number of parameters present in a regular neural network.

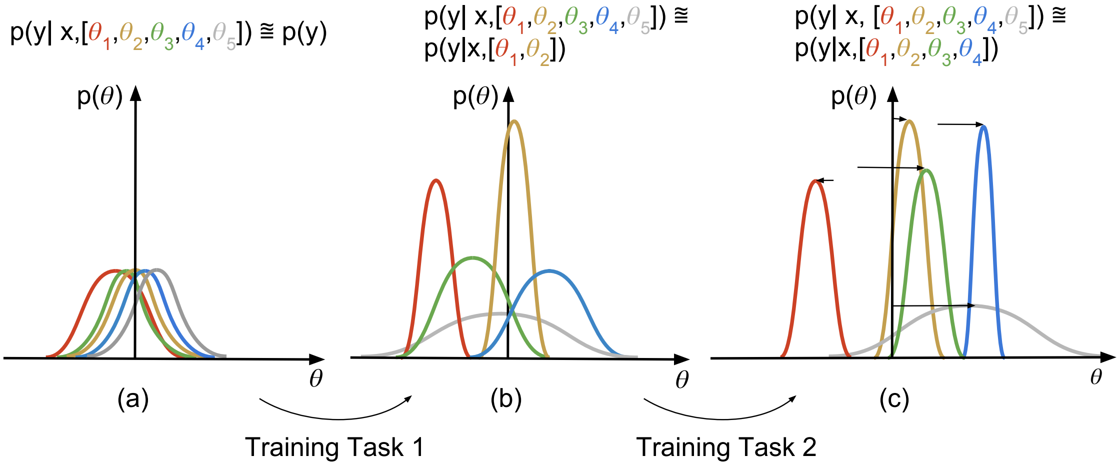


Figure 3.1: Illustration of the evolution of weight distributions – uncertain weights adapt more quickly – when learning two tasks using UCB. (a) weight parameter initialized by distributions initialized with mean and variance values randomly sampled from $\mathcal{N}(0, 0.1)$. (b) posterior distribution after learning task one; while θ_1 and θ_2 exhibit lower uncertainties after learning the first task, θ_3, θ_4, and θ_5 have larger uncertainties, making them available to learn more tasks. (c) a second task is learned using higher learning rates for previously uncertain parameters (θ_1, θ_2, θ_3, and θ_4) while learning rates for θ_1 and θ_2 are reduced. Size of the arrows indicate the magnitude of the change of the distribution mean upon gradient update.

We propose to use the predicted mean and variance of the latent distributions to characterize the importance of each parameter. We perform continual learning with Bayesian neural networks by controlling the learning rate of each parameter as a function of its uncertainty. Figure 3.1 illustrates how posterior distributions evolve for certain and uncertain weight distributions while learning two consecutive tasks. Intuitively, the more uncertain a parameter is, the more learnable it can be and therefore, larger gradient steps can be taken for it to learn the current task. As a hard version of this regularization technique, we also show that pruning, i.e., preventing the most important model parameters from any change and learning new tasks with the remaining parameters, can be also integrated into UCB. We refer to this method as UCB-P.

Our main contirbutions are as follows: We propose to perform continual learning with Bayesian neural networks and develop a new method which exploits the inherent measure of uncertainty therein to adapt the learning rate of individual parameters (Sec. 3.4). Second, we introduce a hard-threshold variant of our method that decides which parameters to freeze (Sec. 3.4.2). Third, in Sec. 3.5, we extensively validate our approach experimentally, comparing it to prior art both on single datasets split into different tasks, as well as for the more difficult scenario of learning a sequence of different datasets. Forth, in contrast to most prior work, our approach does not rely on knowledge about task boundaries at inference time, which humans do not need and might not be always available. We show in Sec. 3.6 that our approach naturally supports this scenario and does not require task information at test time, sometimes also referred to as a “single head” scenario for all tasks. We refer to evaluation metric of a “single head" model without task information at test time as “generalized accuracy".

3.2 Bayesian Approaches for Continual Learning

Using Bayesian approach in learning neural networks has been studied for few decades (MacKay, 1992b; MacKay, 1992a). Several approaches have been proposed for Bayesian neural networks, based on, e.g., the Laplace approximation (MacKay, 1992a), Hamiltonian Monte Carlo (Neal, 2012), variational inference (Hinton & Van Camp, 1993; Graves, 2011), and probabilistic backpropagation (Hernández-Lobato & Adams, 2015). Variational continual learning (Nguyen *et al.,* 2018) uses Bayesian inference to perform continual learning where new posterior distribution is simply obtained by multiplying the previous posterior by the likelihood of the dataset belonging to the new task. They also showed that by using a core-set, a small representative set of data from previous tasks, VCL can experience less forgetting. In contrast, we rely on Bayesian neural networks to use their predictive uncertainty to perform continual learning. Moreover, we do not use episodic memory or any other way to access or store previous data in our approach.

3.2.0.1 Natural gradient descent methods:

A fast natural gradient descent method for variational inference was introduced in (Khan & Nielsen, 2018) in which, the Fisher Information matrix is approximated using the generalized Gauss-Newton method. In contrast, in our work, we use classic gradient descent. Although second order optimization algorithms are proven to be more accurate than the first order methods, they add considerable computational cost. (Tseran *et al.,* 2018; Chen *et al.,* 2019) both investigate the effect of natural gradient descent methods as an alternative to classic gradient descent used in VCL and EWC methods.

GNG (Chen *et al.,* 2019) uses Gaussian natural gradients in the Adam optimizer (Kingma & Ba, 2015) in the framework of VCL because as opposed to conventional gradient methods which perform in Euclidian space, natural gradients cause a small difference in terms of distributions following the changes in parameters in the Riemannian space. Similar to VCL, they obtained their best performance by adding a coreset of previous examples. (Tseran *et al.,* 2018) introduce two modifications to VCL called Natural-VCL (N-VCL) and VCL-Vadam. N-VCL (Tseran *et al.,* 2018) uses a Gauss-Newton approximation introduced by (Schraudolph, 2002; Graves, 2011) to estimate the VCL objective function and used natural gradient method proposed in (Khan *et al.,* 2018) to exploit the Riemannian geometry of the variational posterior by scaling the gradient with an adaptive learning rate equal to σ^{-2} obtained by approximating the Fisher Information matrix in an online fashion. VCL-Vadam

(Tseran *et al.,* 2018) is a simpler version of N-VCL to trade-off accuracy for simplicity which uses Vadam (Khan *et al.,* 2018) to update the gradients by perturbing the weights with a Gaussian noise using a reparameterization trick and scaling by σ^{-1} instead of its squared. N-VCL/VCL-Vadam both use variational inference to adapt the learning rate within Adam optimizer at every time step, whereas in our method below, gradient decent is used with constant learning rate during each task where learning rate scales with uncertainty only after finishing a task. We show extensive comparison with state-of-the-art results on short and relatively long sequence of vision datasets with Bayesian *convolutional* neural networks, whereas VCL-Vadam only rely on multi-layer perceptron networks. We also like to highlight that this is the first work which evaluates and shows the working of convolutional Bayesian Neural Networks rather than only fully connected MLP models for continual learning.

3.3 Variational Bayes-by-Backprop (BBB)

In this section, we review the Bayes-by-Backprop (BBB) framework which was introduced by (Blundell *et al.,* 2015); to learn a probability distribution over network parameters. (Blundell *et al.,* 2015) showed a back-propagation-compatible algorithm which acts as a regularizer and yields comparable performance to dropout on the MNIST dataset. In Bayesian models, latent variables are drawn from a prior density $p(\mathbf{w})$ which are related to the observations through the likelihood $p(\mathbf{x}|\mathbf{w})$. During inference, the posterior distribution $p(\mathbf{w}|\mathbf{x})$ is computed conditioned on the given input data. However, in practice, this probability distribution is intractable and is often estimated through approximate inference. Markov Chain Monte Carlo (MCMC) sampling (Hastings, 1970) has been widely used and explored for this purpose, see (Robert & Casella, 2013) for different methods under this category. However, MCMC algorithms, despite providing guarantees for finding asymptotically exact samples from the target distribution, are not suitable for large datasets and/or large models as they are bounded by speed and scalability issues. Alternatively, variational inference provides a faster solution to the same problem in which the posterior is approximated using optimization rather than being sampled from a chain (Hinton & Van Camp, 1993). Variational inference methods always take advantage of fast optimization techniques such as stochastic methods or distributed methods, which allow them to explore data models quickly. See (Blei *et al.,* 2017) for a complete review of the theory and (Shridhar *et al.,* 2018) for more discussion on how to use Bayes by Backprop (BBB) in convolutioal neural networks.

Let $\mathbf{x} \in \mathbb{R}^n$ be a set of observed variables and $\mathbf{w}$ be a set of latent variables. A neural network, as a probabilistic model $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$, given a set of training examples $\mathcal{D} = (\mathbf{x}, \mathbf{y})$ can output $\mathbf{y}$ which belongs to a set of classes by using the set of weight parameters $\mathbf{w}$. Variational inference aims to calculate this conditional probability distribution over the latent variables by finding the closest proxy to the exact posterior by solving an optimization problem.

We first assume a family of probability densities over the latent variables $\mathbf{w}$ parametrized by θ, i.e., $q(\mathbf{w}|\theta)$. We then find the closest member of this family to the true conditional probability of interest $P(\mathbf{w}|\mathcal{D})$ by minimizing the Kullback-Leibler (KL) divergence between

q and P which is equivalent to minimizing variational free energy or maximizing the expected lower bound:

$$\theta^* = \arg\min_\theta \mathrm{KL}\big(q(\mathbf{w}|\theta) \| P(\mathbf{w}|\mathcal{D})\big) \tag{3.1}$$

The objective function can be written as:

$$\mathcal{L}_{BBB}(\theta, \mathcal{D}) = \mathrm{KL}\big[q(\mathbf{w}|\theta) \| P(\mathbf{w})\big] - \mathbb{E}_{q(\mathbf{w}|\theta)}\big[\log(P(\mathcal{D}|\mathbf{w}))\big] \tag{3.2}$$

Eq. 3.2 can be approximated using N Monte Carlo samples $\mathbf{w}_i$ from the variational posterior (Blundell *et al.*, 2015):

$$\mathcal{L}_{BBB}(\theta, \mathcal{D}) \approx \sum_{i=1}^{N} \log q(\mathbf{w}_i|\theta) - \log P(\mathbf{w}_i) - \log(P(\mathcal{D}|\mathbf{w}_i)) \tag{3.3}$$

We assume $q(\mathbf{w}|\theta)$ to have a Gaussian pdf with diagonal covariance and parametrized by $\theta = (\mu, \rho)$. A sample weight of the variational posterior can be obtained by sampling from a unit Gaussian and reparametrized by $\mathbf{w} = \mu + \sigma \circ \epsilon$ where ϵ is the noise drawn from unit Gaussian, and $\circ$ is a pointwise multipliation. Standard deviation is parametrized as $\sigma = \log(1 + \exp(\rho))$ and thus is always positive. For the prior, as suggested by (Blundell *et al.*, 2015), a scale mixture of two Gaussian pdfs are chosen which are zero-centered while having different variances of σ_1^2 and σ_2^2. The uncertainty obtained for every parameter has been successfully used in model compression (Han *et al.*, 2015) and uncertainty-based exploration in reinforcement learning (Blundell *et al.*, 2015). In this work we propose to use this framework to learn sequential tasks without forgetting using per-weight uncertainties.

3.4 Uncertainty-guided Continual Learning in Bayesian Neural Networks

In this section, we introduce Uncertainty-guided Continual learning approach with Bayesian neural networks (UCB), which exploits the estimated uncertainty of the parameters' posterior distribution to regulate the change in "important" parameters both in a soft way (Section 3.4.1) or setting a hard threshold (Section 3.4.2).

3.4.1 UCB with learning rate regularization

A common strategy to perform continual learning is to reduce forgetting by regularizing further changes in the model representation based on parameters' *importance*. In UCB the regularization is performed with the learning rate such that the learning rate of each parameter and hence its gradient update becomes a function of its *importance*.

As shown in the following equations, in particular, we scale the learning rate of μ and ρ for each parameter distribution inversely proportional to its importance Ω to reduce changes in important parameters while allowing less important parameters to alter more in favor of learning new tasks.

$$\alpha_\mu \leftarrow {}^{\alpha_\mu}/_{\Omega_\mu} \tag{3.4}$$

$$\alpha_\rho \leftarrow {}^{\alpha_\rho}/_{\Omega_\rho} \tag{3.5}$$

The core idea of this work is to base the definition of importance on the well-defined uncertainty in parameters distribution of Bayesian neural networks, i.e., setting the *importance* to be inversely proportional to the standard deviation σ which represents the parameter uncertainty in the Bayesian neural network:

$$\Omega \propto {}^{1}/_{\sigma} \tag{3.6}$$

We explore different options to set Ω in our ablation study presented in Table 3.5. We empirically found that $\Omega_\mu = 1/\sigma$ and not adapting the learning rate for ρ (i.e. $\Omega_\rho = 1$) yields the highest accuracy and the least forgetting.

The key benefit of UCB with learning rate as the regularizer is that it neither requires additional memory, as opposed to pruning technique nor tracking the change in parameters with respect to the previously learned task, as needed in common weight regularization methods.

More importantly, this method does not need to be aware of task switching as it only needs to adjust the learning rates of the means in the posterior distribution based on their current uncertainty. The complete algorithm for UCB is shown in Algorithm 2 with parameter update function given in Algorithm 3.

3.4.2 UCB using weight pruning (UCB-P)

In this section, we introduce a variant of our method, UCB-P, which is related to recent efforts in weight pruning in the context of reducing inference computation and network compression (Liu *et al.,* 2017; Molchanov *et al.,* 2016). More specifically, weight pruning has been recently used in continual learning (Mallya & Lazebnik, 2018), where the goal is to continue learning multiple tasks using a single network's capacity. (Mallya & Lazebnik, 2018) accomplished this by freeing up parameters deemed to be unimportant to the current task according to their magnitude. Forgetting is prevented in pruning by saving a task-specific binary mask of important vs. unimportant parameters. Here, we adapt pruning to Bayesian neural networks. Specifically, we propose a different criterion for measuring importance: the statistically-grounded uncertainty defined in Bayesian neural networks.

Unlike regular deep neural networks, in a BBB model weight parameters are represented by probability distributions parametrized by their mean and standard deviation. Similar to (Blundell *et al.,* 2015), in order to take into account both mean and standard deviation, we use the signal-to-noise ratio (SNR) for each parameter defined as

$$\Omega = \text{SNR} = {}^{|\mu|}/_{\sigma} \tag{3.7}$$

Algorithm 2 Uncertainty-guided Continual Learning with Bayesian Neural Networks UCB

```
Require Training data for all tasks $\mathcal{D} = (\mathbf{x}, \mathbf{y})$, $\mu$ (mean of posterior), $\rho$, $\sigma_1$ and $\sigma_2$ (std for the scaled mixture Gaussian pdf of prior), $\pi$ (weighting factor for prior), $N$ (number of samples in a mini-batch), $M$ (Number of minibatches per epoch), initial learning rate ($\alpha_0$)
$\alpha_\mu = \alpha_\rho = \alpha_0$
for every task do
    repeat
        $\epsilon \sim \mathcal{N}(0, I)$
        $\sigma = \log(1 + \exp(\rho))$    ▷ Ensures $\sigma$ is always positive
        $\mathbf{w} = \mu + \sigma \circ \epsilon$    ▷$\mathbf{w} = \{\mathbf{w}_1, \ldots, \mathbf{w}_i, \ldots, \mathbf{w}_N\}$ posterior samples of weights
        $l_1 = \sum_{i=1}^{N} \log \mathcal{N}(\mathbf{w}_i|\mu, \sigma^2)$    ▷ $l_1$ := Log-posterior
        $l_2 = \sum_{i=1}^{N} \log \left(\pi\mathcal{N}(\mathbf{w}_i \mid 0, \sigma_1^2) + (1-\pi)\mathcal{N}(\mathbf{w}_i \mid 0, \sigma_2^2)\right)$    ▷ $l_2$ := Log-prior
        $l_3 = \sum_{i=1}^{N} \log(p(\mathcal{D}|\mathbf{w}_i))$    ▷ $l_3$ := Log-likelihood of data
        $\mathcal{L}_{BBB} = \frac{1}{M}(l_1 - l_2 - l_3\ )$
        $\mu \leftarrow \mu - \alpha_\mu \nabla \mathcal{L}_{BBB_\mu}$
        $\rho \leftarrow \rho - \alpha_\rho \nabla \mathcal{L}_{BBB_\rho}$
    until loss plateaus
    $\alpha_\mu, \alpha_\rho \leftarrow$ LearningRateUpdate($\alpha_\mu, \alpha_\rho, \sigma, \mu$)    ▷ See Algorithm 3 for UCB and 4 for UCB-P
```

Algorithm 3 LearningRateUpdate in UCB

```
Given: $\alpha_\mu$, $\alpha_\rho$, $\sigma$
for each parameter do
    $\Omega_\mu \leftarrow 1/\sigma$
    $\Omega_\rho \leftarrow 1$
    $\alpha_\mu \leftarrow \alpha_\mu/\Omega_\mu$
    $\alpha_\rho \leftarrow \alpha_\rho/\Omega_\rho$
```

Algorithm 4 LearningRateUpdate in UCB-P

```
Given: $\alpha_\mu$, $\alpha_\rho$, $\sigma_\mu$
for each parameter $j$ in each layer $l$ do
    $\Omega \leftarrow {}^{|\mu|}/_{\sigma}$    ▷ Signal to noise ratio
    if $\Omega[j] \in$ top $p\%$ of $\Omega$s in $l$ then
        $\alpha_\mu = \alpha_\rho = 0$
```

SNR is a commonly used measure in signal processing to distinguish between "useful" information from unwanted noise contained in a signal. In the context of neural models, the SNR can be thought as an indicative of parameter importance; the higher the SNR, the more effective or important the parameter is to the model predictions for a given task.

UCB-P, as shown in Algorithms 2 and 4, is performed as follows: for every layer, convolutional or fully-connected, the parameters are ordered by their SNR value and those with the lowest importance are pruned (set to zero). The pruned parameters are marked using

Table 3.1: Utilized datasets summary

Names	#Classes	Train	Test
FaceScrub (Ng & Winkler, 2014)	100	20,600	2,289
MNIST (LeCun *et al.*, 1998)	10	60,000	10,000
CIFAR100 (Krizhevsky & Hinton, 2009)	100	50,000	10,000
NotMNIST (Bulatov, 2011)	10	16,853	1,873
SVHN (Netzer *et al.*, 2011)	10	73,257	26,032
CIFAR10 (Krizhevsky & Hinton, 2009)	10	39,209	12,630
TrafficSigns (Stallkamp *et al.*, 2011)	43	39,209	12,630
FashionMNIST (Xiao *et al.*, 2017)	10	60,000	10,000

a binary mask so that they can be used later in learning new tasks whereas the important parameters remain fixed throughout training on future tasks. Once a task is learned, an associated binary mask is saved which will be used during inference to recover key parameters and hence the exact performance to the desired task.

The overhead memory per parameter in encoding the mask as well as saving it on the disk is as follows. Assuming we have n tasks to learn using a single network, the total number of required bits to encode an accumulated mask for a parameter is at max $\log_2 n$ bits assuming a parameter deemed to be important from task 1 and kept being encoded in the mask.

3.5 Experimental Setup

Datasets: We evaluate our approach in two common scenarios for continual learning: 1) class-incremental learning of a single or two randomly alternating datasets, where each task covers only a subset of the classes in a dataset, and 2) continual learning of multiple datasets, where each task is a dataset. We use Split MNIST with 5 tasks (5-Split MNIST) similar to (Nguyen *et al.*, 2018; Chen *et al.*, 2019; Tseran *et al.*, 2018) and permuted MNIST (Srivastava *et al.*, 2013) for class incremental learning with similar experimental settings as used in (Serra *et al.*, 2018; Tseran *et al.*, 2018). Furthermore, to have a better understanding of our method, we evaluate our approach on continually learning a sequence of 8 datasets with different distributions using the identical sequence as in (Serra *et al.*, 2018), which includes FaceScrub (Ng & Winkler, 2014), MNIST, CIFAR100, NotMNIST (Bulatov, 2011), SVHN (Netzer *et al.*, 2011), CIFAR10, TrafficSigns (Stallkamp *et al.*, 2011), and FashionMNIST (Xiao *et al.*, 2017). Table 3.1 shows a summary of the datasets utilized in our work along with their size and number of classes. In all the experiments we resized images to $32 \times 32 \times 3$ if necessary. For datasets with monochromatic images, we replicate the image across all RGB channels. No data augmentation of any kind has been used in our analysis.

Baselines: Within the Bayesian framework, we compare to three models which do not incorporate the importance of parameters, namely fine-tuning, feature extraction, and joint training. In fine-tuning (BBB-FT), training continues upon arrival of new tasks without any forgetting avoidance strategy. Feature extraction, denoted as (BBB-FE), refers to freezing all layers in the network after training the first task and training only the last layer for the remaining tasks. In joint training (BBB-JT) we learn all the tasks jointly in a multitask learning fashion which serves as the upper bound for average accuracy on all tasks, as it does not adhere to the continual learning scenario. We also perform the counterparts for FT, FE, and JT using ordinary neural networks and denote them as ORD-FT, ORD-FE, and ORD-JT. From the prior work, we compare with state-of-the-art approaches including Elastic Weight Consolidation (EWC) (Kirkpatrick *et al.,* 2017), Incremental Moment Matching (IMM) (Lee *et al.,* 2017), Learning Without Forgetting (LWF) (Li & Hoiem, 2016), Less-Forgetting Learning (LFL) (Jung *et al.,* 2016), PathNet (Fernando *et al.,* 2017), Progressive neural networks (PNNs) (Rusu *et al.,* 2016), and Hard Attention Mask (HAT) (Serra *et al.,* 2018) using implementations provided by (Serra *et al.,* 2018). On Permuted MNIST results for SI (Zenke *et al.,* 2017) are reported from (Serra *et al.,* 2018). On Split and Permuted MNIST, results for VCL (Nguyen *et al.,* 2018) are obtained using their original provided code whereas for VCL-GNG (Chen *et al.,* 2019) and VCL-Vadam (Tseran *et al.,* 2018) results are reported from the original work without re-implementation. Because our method lies into the regularization-based regime, we only compare against baselines which do not benefit from episodic or coreset memory.

Hyperparameter tuning: Unlike commonly used tuning techniques which use a validation set composed of *all* classes in the dataset, we only rely on the first two task and their validations set, similar to the setup in (Chaudhry *et al.,* 2019a). In all our experiments we consider a 0.15 split for the validation set on the first two tasks. After tuning, training starts from the beginning of the sequence. Our scheme is different from (Chaudhry *et al.,* 2019a), where the models are trained on the first (e.g. three) tasks for validation and then training is restarted for the remaining ones and the reported performance is only on the remaining tasks.

Training details: It is important to note that in all our experiments, *no pre-trained model is used*. We used stochastic gradient descent with a batch size of 64 and a learning rate of 0.01, decaying it by a factor of 0.3 once the loss plateaued. Dataset splits and batch shuffle are identically in all UCB experiments and all baselines.

Bayes-by-backprop (BBB) Hyperparamters: Table 3.2 shows the search space for hyperparamters in the BBB algorithm (Blundell *et al.,* 2015) which we used for tuning on the validation set of the first two tasks.

Network architecture: For Split MNIST and Permuted MNIST experiments, we have used a two-layer perceptron which has 1200 units. Because there is more number of parameters in our Bayesian neural network compared to its equivalent regular neural net, we

Table 3.2: Search space for hyperparamters in BBB given by (Blundell *et al.*, 2015)

BBB hyperparamters	$-\log \sigma_1$	$-\log \sigma_2$	π
Search space	$\{0, 1, 2\}$	$\{6, 7, 8\}$	{0.25,0.5,0.75}

Table 3.3: Continually learning on CIFAR10/100 using AlexNet and ResNet18 for UCB (our method) and HAT (Serra *et al.*, 2018). BWT and ACC in %. All results are (re)produced by us.

Method	BWT	ACC
HAT (AlexNet)	0.0	78.3
HAT (ResNet18)	−9.0	56.8
UCB (AlexNet)	−0.7	79.44
UCB (ResNet18)	−0.7	79.70

ensured fair comparison by matching the total number of parameters between the two to be 1.9M unless otherwise is stated. For the multiple datasets learning scenario, as well as alternating incremental CIFAR10/100 datasets, we have used a ResNet18 Bayesian neural network with 7.1-11.3M parameters depending on the experiment. However, the majority of the baselines provided in this work are originally developed using some variants of AlexNet structure and altering that, e.g. to ResNet18, resulted in degrading in their reported and experimented performance as shown in Table 3.3. Therefore, we kept the architecture for baselines as AlexNet and ours as ResNet18 and only matched their number of parameters to ensure having equal capacity across different approaches.

Number of Monte Carlo samples: UCB is ensured to be robust to random noise using multiple samples drawn from posteriors. Here we explore different number of samples and the effect on final performance for ACC and BWT. We have used $\Omega_\mu = 1/\sigma$ as importance and regularization has been performed on mean values only. Following the result in Table 3.4 we chose the number of samples to be 10 for all experiments.

Pruning procedure and mask size: Once a task is learned, we compute the performance drop for a set of arbitrary pruning percentages from the maximum training accuracy achieved when no pruning is applied. The pruning portion is then chosen using a threshold beyond which the performance drop is not accepted. Mask size is chosen without having the knowledge of how many tasks to learn in the future. Upon learning each task we used a uniform distribution of pruning ratios (50-100%) and picked the ratio resulted in at most 1%, 2%, and 3% forgetting for MNIST, CIFAR, and 8tasks experiments, respectively. We

Table 3.4: Number of Monte Carlo samples (N) in 2-Split MNIST

Method	N	BWT (%)	ACC (%)
UCB	1	0.00	98.0
UCB	2	0.00	98.3
UCB	5	-0.15	99.0
UCB	10	0.00	99.2
UCB	15	-0.01	98.3

Table 3.5: Variants of learning rate regularization and importance measurement on 2-Split MNIST

Method	μ	ρ	Importance Ω	BWT (%)	ACC (%)
UCB	x	-	$1/\sigma$	0.00	99.2
UCB	-	x	$1/\sigma$	-0.04	98.7
UCB	x	x	$1/\sigma$	-0.02	98.0
UCB	x	-	$\|\mu\|/\sigma$	-0.03	98.4
UCB	-	x	$\|\mu\|/\sigma$	-0.52	98.7
UCB	x	x	$\|\mu\|/\sigma$	-0.32	98.8
UCB-P	x	x	$\|\mu\|/\sigma$	-0.01	99.0
UCB-P	x	x	$1/\sigma$	-0.01	98.9

did not tune this parameter because in our hyperparameter tuning, we only assume we have validation sets of the first two tasks.

Parameter regularization and importance measurement: Table 3.5 ablates different ways to compute the *importance* Ω of an parameter in Eq. 3.4 and 3.5. As shown in Table 3.5 the configuration that yields the highest accuracy and the least forgetting (maximum BWT) occurs when the learning rate regularization is performed only on μ of the posteriors using $\Omega_\mu = 1/\sigma$ as the importance and $\Omega_\rho = 1$.

Performance measurement: Let n be the total number of tasks. Once all are learned, we evaluate our model on all n tasks. ACC is the average test classification accuracy across all tasks. To measure forgetting we report backward transfer, BWT, which indicates how much learning new tasks has influenced the performance on previous tasks. While BWT < 0 directly reports *catastrophic forgetting*, BWT > 0 indicates that learning new tasks has helped with the preceding tasks. Formally, BWT and ACC are as follows:

$$\mathrm{BWT} = \frac{1}{n}\sum_{i=1}^{n} R_{i,n} - R_{i,i}, \quad \mathrm{ACC} = \frac{1}{n}\sum_{i=1}^{n} R_{i,n} \tag{3.8}$$

where $R_{i,n}$ is the test classification accuracy on task i after sequentially finishing learning the n^{th} task. Note that in UCB-P, $R_{i,i}$ refers the test accuracy on task i before pruning and $R_{i,n}$ after pruning which is equivalent to the end of sequence performance. In Section 3.6, we show that our UCB model can be used when tasks labels are not available at inference time by training it with a "single head" architecture with a sum of number of classes for all tasks. We refer to the ACC measured for this scenario as "Generalized Accuracy".

3.5.1 5-Split MNIST

We first present our results for class incremental learning of MNIST (5-Split MNIST) in which we learn the digits $0 - 9$ in five tasks with 2 classes at a time in 5 pairs of 0/1, 2/3, 4/5, 6/7, and 8/9. Table 3.6 shows the results for reference baselines in Bayesian and non-Bayesian neural networks including fine-tuning (BBB-FT, ORD-FT), feature extraction (BBB-FE, ORD-FE) and, joint training (BBB-JT, ORD-JT) averaged over 3 runs and standard deviations are given in Table 3.6. Although the MNIST dataset is an "easy" dataset, we observe throughout all experiments that Bayesian fine-tuning and joint training perform significantly better than their counterparts, ORD-FT and ORD-JT. For Bayesian methods, we compare against VCL and its variations named as VCL with Variational Adam (VCL-Vadam), VCL with Adam and Gaussian natural gradients (VCL-GNG). For non-Bayesian methods, we compare against HAT, IMM, and EWC (EWC can be regarded as Bayesian-inspired). VCL-Vadam (ACC=99.17%) appears to be outperforming VCL (ACC=98.20%) and VCL-GNG (ACC=96.50%) in average accuracy. However, full comparison is not possible because forgetting was not reported for Vadam and GNG. Nevertheless, UCB (ACC=99.63%) is able to surpass all the baselines including VCL-Vadam in average accuracy while in zero forgetting it is on par with HAT (ACC=99.59%). We also report results on incrementally learning MNIST in two tasks (2-Split MNIST) in Table 3.7, where we compare it against PackNet, HAT, and LWF where PackNet, HAT, UCB-P, and UCB have zero forgetting while UCB has marginally higher accuracy than all others.

3.5.2 Permuted MNIST

Permuted MNIST is a popular variant of the MNIST dataset to evaluate continual learning approaches in which each task is considered as a random permutation of the original MNIST pixels. Following the literature, we learn a sequence of 10 random permutations and report average accuracy at the end. Table 3.8 shows ACC and BWT of UCB and UCB-P in comparison to state-of-the-art models using a small and a large network with 0.1M and 1.9M parameters, respectively. The accuracy achieved by UCB (ACC=91.44 ±

Table 3.6: Continually learning on 5-Split MNIST. BWT and ACC in %. (*) denotes that methods do not adhere to the continual learning setup: BBB-JT and ORD-JT serve as the upper bound for ACC for BBB/ORD networks, respectively. All results are (re)produced by us.

Method	BWT	ACC
VCL-Vadam (Tseran *et al.*, 2018)	-	99.17 ± 0.05
VCL-GNG (Chen *et al.*, 2019)	-	96.50 ± 0.07
VCL (Nguyen *et al.*, 2018)	-0.56 ± 0.03	98.20 ± 0.03
IMM (Lee *et al.*, 2017)	-11.20 ± 1.57	88.54 ± 1.56
EWC (Kirkpatrick *et al.*, 2017)	-4.20 ± 1.08	95.78 ± 1.08
HAT (Serra *et al.*, 2018)	0.00 ± 0.02	99.59 ± 0.02
ORD-FT*	-9.18 ± 1.12	90.60 ± 1.12
ORD-FE*	0.00 ± 1.56	98.54 ± 1.57
BBB-FT*	-6.45 ± 1.99	93.42 ± 1.98
BBB-FE*	0.00 ± 2.23	98.76 ± 2.23
UCB-P (Ours)	-0.72 ± 0.04	99.32 ± 0.04
UCB (Ours)	0.00 ± 0.04	$\mathbf{99.63 \pm 0.03}$
ORD-JT*	0.00 ± 0.02	99.78 ± 0.02
BBB-JT*	0.00 ± 0.01	99.87 ± 0.01

0.04%) using the small network outperforms the ACC reported by (Serra *et al.*, 2018) for SI (ACC=86.0%), EWC (ACC=88.2%), while HAT attains a slightly better performance (ACC=91.6%). Comparing the average accuracy reported in VCL-Vadam (ACC=86.34%) and VCL-GNG (ACC=90.50%) as well as obtained results for VCL (ACC=88.80%) shows UCB with BWT=(0.03% ± 0.00%) is able to outperform other Bayesian approaches in accuracy while forgetting significantly less compared to VCL with BWT=-7.9%. While we do not experiment with memory in this work, not surprisingly adding memory to most approaches will improve their performance significantly as it allows looking into past tasks. E.g. (Chen *et al.*, 2019) report ACC=94.37% for VCL-GNC when adding a memory of size 200.

Next, we compare the results for the larger network (1.9M). While HAT and UCB have zero forgetting, UCB, reaching ACC=97.42 ± 0.01%, performs better than all baselines including HAT which obtains ACC=97.34 ± 0.05% using 1.9M parameters. We also observe again that BBB-FT, despite being not specifically penalized to prevent forgetting, exhibits reasonable negative BWT values, performing better than IMM and LWF baselines. It is close to joint training, BBB-JT, with ACC=98.1%, which can be seen as an upper bound.

Table 3.7: Continually learning on 2-Split MNIST. BWT and ACC in %. (*) denotes that methods do not adhere to the continual learning setup: BBB-JT and ORD-JT serve as the upper bound for ACC for BBB/ORD networks, respectively. All results are (re)produced by us.

Method	BWT	ACC
PackNet (Mallya & Lazebnik, 2018)	0.04 ± 0.01	98.91 ± 0.03
LWF (Li & Hoiem, 2016)	-0.22 ± 0.04	99.12 ± 0.03
HAT (Serra *et al.*, 2018)	0.01 ± 0.00	99.02 ± 0.00
ORD-FT	-6.81 ± 0.03	92.42 ± 0.02
ORD-FE	0.04 ± 0.04	97.90 ± 0.04
BBB-FT	-0.61 ± 0.03	98.44 ± 0.03
BBB-FE	0.02 ± 0.05	98.03 ± 0.05
UCB-P (Ours)	0.03 ± 0.04	99.02 ± 0.01
UCB (Ours)	0.01 ± 0.00	$\mathbf{99.18 \pm 0.01}$
ORD-JT*	0.02 ± 0.03	99.13 ± 0.03
BBB-JT*	0.03 ± 0.02	99.51 ± 0.02

3.5.3 Alternating CIFAR10 and CIFAR100

In this experiment, we randomly alternate between class incremental learning of CIFAR10 and CIFAR100. Both datasets are divided into 5 tasks each with 2 and 20 classes per task, respectively. Table 3.9 presents ACC and BWT obtained with UCB-P, UCB, and three BBB reference methods compared against various continual learning baselines. Among the baselines presented in Table 3.9, PNN and PathNet are the only zero-forgetting-guaranteed approaches. It is interesting to note that in this setup, some baselines (PathNet, LWF, and LFL) do not perform better than the naive accuracy achieved by feature extraction. PathNet suffers from bad pre-assignment of the network's capacity per task which causes poor performance on the initial task from which it never recovers. IMM performs almost similar to fine-tuning in ACC, yet forgets more. PNN, EWC, and HAT are the only baselines that perform better than BBB-FE and BBB-FT. EWC and HAT are both allowed to forget by construction, however, HAT shows zero forgetting behavior. While EWC is outperformed by both of our UCB variants, HAT exhibits 1% better ACC over UCB-P. Despite having a slightly higher forgetting, the overall accuracy of UCB is higher, reaching 79.4%. BBB-JT in this experiment achieves a positive BWT which shows that learning the entire sequence improves the performance on earlier tasks.

Table 3.8: Continually learning on Permuted MNIST. BWT and ACC in %. (*) denotes that method does not adhere to the continual learning setup: BBB-JT serves as the upper bound for ACC for BBB network. ‡ denotes results reported by (Serra *et al.*, 2018). † denotes the result reported from original work. BWT was not reported in ‡ and †. All others results are (re)produced by us.

Method	#Params	BWT	ACC
SI (Zenke *et al.*, 2017)‡	0.1M	-	86.0
EWC (Kirkpatrick *et al.*, 2017)‡	0.1M	-	88.2
HAT (Serra *et al.*, 2018)‡	0.1M	-	91.6
VCL-Vadam†	0.1M	-	93.34
VCL-GNG†	0.1M	-	94.62
VCL	0.1M	-7.90 ± 0.23	88.80 ± 0.23
UCB (Ours)	0.1M	-0.38 ± 0.02	91.44 ± 0.04
LWF (Li & Hoiem, 2016)	1.9M	-31.17 ± 0.05	65.65 ± 0.05
IMM (Lee *et al.*, 2017)	1.9M	-7.14 ± 0.07	90.51 ± 0.08
HAT (Serra *et al.*, 2018)	1.9M	0.03 ± 0.05	97.34 ± 0.05
BBB-FT	1.9M	-0.58 ± 0.05	90.01 ± 0.05
BBB-FE	1.9M	0.02 ± 0.03	93.54 ± 0.04
UCB-P (Ours)	1.9M	-0.95 ± 0.06	97.24 ± 0.06
UCB (Ours)	1.9M	0.03 ± 0.00	$\mathbf{97.42 \pm 0.01}$
BBB-JT*	1.9M	0.00 ± 0.00	98.12 ± 0.01

3.5.4 Multiple datasets learning

Finally, we present our results for continual learning of 8 tasks using UCB-P and UCB in Table 3.10. Similar to the previous experiments we look at both ACC and BWT obtained for UCB-P, UCB, BBB references (FT, FE, JT) as well as various baselines. Considering the ACC achieved by BBB-FE or BBB-FT (58.1%) as a lower bound we observe again that some baselines are not able to do better than BBB-FT including LFL, PathNet, LWF, IMM, and EWC while PNN and HAT remain the only strong baselines for our UCB-P and UCB approaches. UCB-P again outperforms PNN by 3.6% in ACC. HAT exhibits only −0.1% BWT, but our UCB achieves 2.4% higher ACC.

3.6 Single Head and Generalized Accuracy of UCB

UCB can be used even if the task information is not given at test time. For this purpose, at training time, instead of using a separate fully connected classification head for each task, we use a single head with the total number of outputs for all tasks. For example in the

Table 3.9: Continually learning on CIFAR10/100. BWT and ACC in %. (*) denotes that method does not adhere to the continual learning setup: BBB-JT serves as the upper bound for ACC for BBB network. All results are (re)produced by us.

Method	BWT	ACC
PathNet (Fernando *et al.*, 2017)	0.00 ± 0.00	28.94 ± 0.03
LWF (Li & Hoiem, 2016)	-37.9 ± 0.32	42.93 ± 0.30
LFL (Jung *et al.*, 2016)	-24.22 ± 0.21	47.67 ± 0.22
IMM (Lee *et al.*, 2017)	-12.23 ± 0.06	69.37 ± 0.06
PNN (Rusu *et al.*, 2016)	0.00 ± 0.00	70.73 ± 0.08
EWC (Kirkpatrick *et al.*, 2017)	-1.53 ± 0.07	72.46 ± 0.06
HAT (Serra *et al.*, 2018)	0.04 ± 0.06	78.32 ± 0.06
BBB-FE	0.04 ± 0.02	51.04 ± 0.03
BBB-FT	-7.43 ± 0.07	68.89 ± 0.07
UCB-P (Ours)	-1.89 ± 0.03	77.32 ± 0.03
UCB (Ours)	-0.72 ± 0.02	$\mathbf{79.44 \pm 0.02}$
BBB-JT*	1.52 ± 0.04	83.93 ± 0.04

8-dataset experiment we only use one head with 293 number of output classes, rather than using 8 separate heads, during training and inference time.

Table 3.11 presents our results for UCB and BBB-FT trained with a single head against having a multi-head architecture, in columns 4-7. Interestingly, we see only a small performance degrade for UCB from training with multi-head to a single head. The ACC reduction is 0.3%, 2.6%, 5.1%, and 4.1% for 2-Split MNIST, Permuted MNIST, Alternating CIFAR10/100, and sequence of 8 tasks experiments, respectively.

We evaluated UCB and BBB-FT with a more challenging metric where the prediction space covers the classes across all the tasks. Hence, confusion of similar class labels across tasks can be measured. Performance for this condition is reported as Generalized ACC in Table 3.11 in columns 2-3. We observe a small performance reduction in going from ACC to Generalized ACC, suggesting non-significant confusion caused by the presence of more number of classes at test time. The performance degradation from ACC to Generalized ACC is 0.2%, 2.6%, 3.1%, and 3.1% for 2-Split MNIST, Permuted MNIST, Alternating CIFAR10/100, and sequence of 8 tasks, respectively. This shows that UCB can perform competitively in more realistic conditions such as unavailability of task information at test time. We believe the main insight of our approach is that instead of computing additional measurements of importance, which are often task, input or output dependent, we directly use predicted weight uncertainty to find important parameters. We can freeze them using a binary mask, as in UCB-P, or regularize changes conditioned on current uncertainty, as in UCB.

Table 3.10: Continually learning on sequence of 8 datastes. BWT and ACC in %. (*) denotes that method does not adhere to the continual learning setup: BBB-JT serves as the upper bound for ACC for BBB network. All results are (re)produced by us.

Method	BWT	ACC
LFL	-10.0	8.61
PathNet	0.00	20.22
LWF	-54.3	28.22
IMM	-38.5	43.93
EWC	-18.04	50.68
PNN	0.00	76.78
HAT	-0.14	81.59
BBB-FT	-23.1	43.09
BBB-FE	-0.01	58.07
UCB-P (Ours)	-2.54	80.38
UCB (Ours)	-0.84	**84.04**
BBB-JT*	-1.2	84.1

Table 3.11: Single Head vs. Multi-Head architecture and Generalized vs. Standard Accuracy. Generalized accuracy means that task information is not available at test time. SM, PM, CF, and 8T denote the 5-Split MNIST, Permuted MNIST, Alternating CIFAR10/100, and sequence of 8 tasks, respectively.

	Generalized ACC		ACC			
	Single Head		Single Head		Multi Head	
Exp	UCB	BBB-FT	UCB	BBB-FT	UCB	BBB-FT
SM	98.7	98.1	98.9	98.7	99.2	98.4
PM	92.5	86.1	95.1	88.3	97.7	90.0
CF	71.2	65.2	74.3	67.8	79.4	68.9
8T	76.8	47.6	79.9	53.2	84.0	43.1

3.7 Summary

In this chapter, we proposed a continual learning formulation with Bayesian neural networks, called UCB, that uses per-parameter uncertainty predictions to guide the continual learning agent. We showed how the important parameters can be either fully preserved through a saved binary mask (UCB-P) or allowed to change conditioned on their uncertainty for learning new tasks (UCB). We demonstrated the effectiveness of UCB in continually learning short and long sequences of benchmark datasets compared against baselines and prior work. We also showed how UCB can also be deployed as a task-free algorithm by using a single head setting where tasks information is not available at test time.

Chapter 4

Conclusion and Future Work

4.1 Discussion of Contributions

This thesis aimed to explore continual learning in two different neural networks fomulation: Bayesian and non-Bayesian networks.. In the first part, we mainly focused on formulating continual learning in a general setting where we used architecture growth and memory replay to prevent forgetting. In the second part, we developed a regularization-based model in Bayesian neural networks guided by the uncertainty defined per parameters. Summary of our key contributions is provided as follow:

- Chapter 2: we propose a novel hybrid continual learning algorithm – Adversarial Continual Learning (ACL)– that factorizes the representation learned for a sequence of tasks into *task-specific* and *task-invariant* features where the former is important to be fully preserved to avoid forgetting and the latter is empirically found to be remarkably less prone to forgetting. The novelty of our work is that we use adversarial learning along with orthogonality constraints to disentangle the shared and private latent representations which results in compact private modules that can be stored into memory and hence, efficiently preventing forgetting. A tiny replay buffer, although not critical, can be also integrated into our approach if forgetting occurs in the shared module. We evaluated ACL on CL benchmark datasets and established a new state of the art on 20-Split miniImageNet, 5-Datasets, 20-Split CIFAR100, Permuted MNIST, and 5-Split MNIST.

- Chapter 3: we propose a continual learning formulation with Bayesian neural networks, called UCB, that uses uncertainty predictions to perform continual learning: important parameters can be either fully preserved through a saved binary mask (UCB-P) or allowed to change conditioned on their uncertainty for learning new tasks (UCB). We demonstrated how the probabilistic uncertainty distributions per weight are helpful to continually learning short and long sequences of benchmark datasets compared against baselines and prior work. We show that UCB performs superior or on par with state-of-the-art models such as HAT (Serra *et al.,* 2018) across all the experiments. Choosing between the two UCB variants depends on the application scenario: While UCB-P enforces no forgetting after the initial pruning stage by saving a small binary mask per task, UCB does not require additional memory and allows for more learning flexibility in the network by allowing small forgetting to occur. UCB can also be used in a single head setting where the right subset of classes belonging to the task is not known during inference leading to a competitive model that can be deployed where it is not possible to distinguish tasks in a continuous stream of the data at test time. UCB can also be deployed in a single head scenario and where tasks information is not available at test time.

4.2 Future Perspectives

Now, we describe some of the future directions that immediately follow from the quantitative and qualitative analysis provided in this work:

- **Directions to expand ACL approach**: ACL is introduced as a task-aware algorithm in which we assume task descriptions are available during test time. The validity of this assumption can be questionable in real-world applications where this information might not be available. Further expanding this approach can focus on overcoming this limitation by using single head technique where the performance might drop in exchange for becoming task-independent.

- **Directions to expand continual learning in Bayesian neural networks:** Despite being strong and mathematically grounded, Bayesian neural networks in general are computationally more expensive than regular neural nets. To facilitate using BNNs one can explore paths to reduce expenses associated with using double number of parameters in such networks. Another direction to enhance BNNs' capabilities in modeling task-free CL would be removing the assumption of clear boundaries between tasks by determining their correlation between the change in parameters' uncertainty during tasks transitions.

Bibliography

1. Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M. & Tuytelaars, T. *Memory aware synapses: Learning what (not) to forget* in *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), 139–154.
2. Azadi, S., Pathak, D., Ebrahimi, S. & Darrell, T. Compositional gan: Learning conditional image composition (2018).
3. Azadi, S., Pathak, D., Ebrahimi, S. & Darrell, T. Compositional GAN: Learning Image-Conditional Binary Composition.
4. Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association* **112,** 859–877 (2017).
5. Blum, A. & Mitchell, T. *Combining labeled and unlabeled data with co-training* in *Proceedings of the eleventh annual conference on Computational learning theory* (1998), 92–100.
6. Blundell, C., Cornebise, J., Kavukcuoglu, K. & Wierstra, D. *Weight Uncertainty in Neural Network* in *Proceedings of the 32nd International Conference on Machine Learning* (eds Bach, F. & Blei, D.) **37** (PMLR, 2015), 1613–1622.
7. Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D. & Erhan, D. *Domain separation networks* in *Advances in neural information processing systems* (2016), 343–351.
8. Bulatov, Y. Notmnist dataset. *Google (Books/OCR), Tech. Rep.[Online]. Available: http://yaroslavvb. blogspot. it/2011/09/notmnist-dataset. html* (2011).
9. Chaudhry, A., Ranzato, M., Rohrbach, M. & Elhoseiny, M. *Efficient Lifelong Learning with A-GEM* in *International Conference on Learning Representations* (2019).
10. Chaudhry, A. *et al.* Continual Learning with Tiny Episodic Memories. *arXiv preprint arXiv:1902.10486* (2019).
11. Chaudhuri, K., Kakade, S. M., Livescu, K. & Sridharan, K. *Multi-view clustering via canonical correlation analysis* in *Proceedings of the 26th annual international conference on machine learning* (2009), 129–136.
12. Chen, Y., Diethe, T. & Lawrence, N. Facilitating Bayesian Continual Learning by Natural Gradients and Stein Gradients. *arXiv preprint arXiv:1904.10644* (2019).

13. Ebrahimi, S., Elhoseiny, M., Darrell, T. & Rohrbach, M. *Uncertainty-guided Continual Learning with Bayesian Neural Networks* in *International Conference on Learning Representations* (2020). https://openreview.net/forum?id=HklUCCVKDB.

14. Ebrahimi, S., Meier, F., Calandra, R., Darrell, T. & Rohrbach, M. Adversarial Continual Learning. *arXiv preprint arXiv:2003.09553* (2020).

15. Ebrahimi, S., Rohrbach, A. & Darrell, T. Gradient-free policy architecture search and adaptation. *arXiv preprint arXiv:1710.05958* (2017).

16. Ebrahimi, S., Rohrbach, A. & Darrell, T. *Gradient-Free Supervised and Unsupervised Learning with Rewards* in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2017).

17. Ebrahimi, S., Sinha, S. & Darrell, T. Variational Adversarial Active Learning.

18. Fernando, C. *et al.* Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734* (2017).

19. Ganin, Y. *et al.* Domain-adversarial training of neural networks. *The Journal of Machine Learning Research* **17,** 2096–2030 (2016).

20. Goodfellow, I. *et al. Generative adversarial nets* in *Advances in neural information processing systems* (2014), 2672–2680.

21. Graves, A. *Practical variational inference for neural networks* in *Advances in neural information processing systems* (2011), 2348–2356.

22. Han, S., Mao, H. & Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

23. Hastings, W. K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* (1970).

24. Hernández-Lobato, J. M. & Adams, R. *Probabilistic backpropagation for scalable learning of bayesian neural networks* in *International Conference on Machine Learning* (2015), 1861–1869.

25. Hinton, G. E. & Van Camp, D. *Keeping the neural networks simple by minimizing the description length of the weights* in *Proceedings of the sixth annual conference on Computational learning theory* (1993), 5–13.

26. Hoffman, J. *et al. CyCADA: Cycle-Consistent Adversarial Domain Adaptation* in *International Conference on Machine Learning* (2018), 1989–1998.

27. Iandola, F. N. *et al.* SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

28. Jung, H., Ju, J., Jung, M. & Kim, J. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122* (2016).

29. Kemker, R. & Kanan, C. *FearNet: Brain-Inspired Model for Incremental Learning* in *International Conference on Learning Representations* (2018). `https://openreview.net/forum?id=SJ1Xmf-Rb`.

30. Khan, M. E. & Nielsen, D. *Fast yet simple natural-gradient descent for variational inference in complex models* in *2018 International Symposium on Information Theory and Its Applications (ISITA)* (2018), 31–35.

31. Khan, M. E. *et al.* Fast and scalable Bayesian deep learning by weight-perturbation in Adam. *arXiv preprint arXiv:1806.04854* (2018).

32. Kim, H. & Mnih, A. Disentangling by factorising. *arXiv preprint arXiv:1802.05983* (2018).

33. Kingma, D. P. & Ba, J. *Adam: A method for stochastic optimization* in *International Conference on Learning Representations* (2015).

34. Kirkpatrick, J. *et al.* Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences,* 201611835 (2017).

35. Krizhevsky, A. & Hinton, G. *Learning multiple layers of features from tiny images* tech. rep. (Citeseer, 2009).

36. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86,** 2278–2324 (1998).

37. Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W. & Zhang, B.-T. *Overcoming catastrophic forgetting by incremental moment matching* in *Advances in Neural Information Processing Systems* (2017), 4652–4662.

38. Li, Z. & Hoiem, D. *Learning Without Forgetting* in *European Conference on Computer Vision* (2016), 614–629.

39. Liu, Z. *et al. Learning efficient convolutional networks through network slimming* in *Proceedings of the IEEE International Conference on Computer Vision* (2017), 2736–2744.

40. Lopez-Paz, D. *et al. Gradient episodic memory for continual learning* in *Advances in Neural Information Processing Systems* (2017), 6467–6476.

41. MacKay, D. J. A practical Bayesian framework for backpropagation networks. *Neural computation* **4,** 448–472 (1992).

42. MacKay, D. J. *Bayesian methods for adaptive models* PhD thesis (California Institute of Technology, 1992).

43. Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I. & Frey, B. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644* (2015).

44. Mallya, A. & Lazebnik, S. *Packnet: Adding multiple tasks to a single network by iterative pruning* in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

45. McClelland, J. L., McNaughton, B. L. & O'reilly, R. C. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review* **102,** 419 (1995).

46. McCloskey, M. & Cohen, N. J. in *Psychology of learning and motivation* 109–165 (Elsevier, 1989).

47. McCloskey, M. & Cohen, N. J. in *Psychology of learning and motivation* 109–165 (Elsevier, 1989).

48. Molchanov, P., Tyree, S., Karras, T., Aila, T. & Kautz, J. *Pruning Convolutional Neural Networks for Resource Efficient Inference* in *International Conference on Learning Representations (ICLR)* (2016).

49. Neal, R. M. *Bayesian learning for neural networks* (Springer Science & Business Media, 2012).

50. Netzer, Y. *et al. Reading digits in natural images with unsupervised feature learning* in *NIPS workshop on deep learning and unsupervised feature learning* (2011).

51. Ng, H.-W. & Winkler, S. *A data-driven approach to cleaning large face datasets* in *Image Processing (ICIP), 2014 IEEE International Conference on* (2014), 343–347.

52. Nguyen, C. V., Li, Y., Bui, T. D. & Turner, R. E. *Variational Continual Learning* in *International Conference on Learning Representations* (2018).

53. Rebuffi, S.-A., Kolesnikov, A., Sperl, G. & Lampert, C. H. *icarl: Incremental classifier and representation learning* in *CVPR* (2017).

54. Riemer, M. *et al.* Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910* (2018).

55. Robert, C. & Casella, G. *Monte Carlo statistical methods* (Springer Science & Business Media, 2013).

56. Robins, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science* **7,** 123–146 (1995).

57. Rusu, A. A. *et al.* Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

58. Salzmann, M., Ek, C. H., Urtasun, R. & Darrell, T. *Factorized orthogonal latent spaces* in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), 701–708.

59. Schönfeld, E., Ebrahimi, S., Sinha, S., Darrell, T. & Akata, Z. Cross-Linked Variational Autoencoders for Generalized Zero-Shot Learning (2019).

60. Schonfeld, E., Ebrahimi, S., Sinha, S., Darrell, T. & Akata, Z. *Generalized zero-and few-shot learning via aligned variational autoencoders* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), 8247–8255.

61. Schonfeld, E., Ebrahimi, S., Sinha, S., Darrell, T. & Akata, Z. *Generalized Zero-Shot Learning via Aligned Variational Autoencoders* in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2019).

62. Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation* **14** (2002).

63. Serra, J., Suris, D., Miron, M. & Karatzoglou, A. *Overcoming Catastrophic Forgetting with Hard Attention to the Task* in *Proceedings of the 35th International Conference on Machine Learning* (eds Dy, J. & Krause, A.) **80** (PMLR, 2018), 4548–4557.

64. Shin, H., Lee, J. K., Kim, J. & Kim, J. *Continual learning with deep generative replay* in *Advances in Neural Information Processing Systems* (2017), 2990–2999.

65. Shon, A., Grochow, K., Hertzmann, A. & Rao, R. P. *Learning shared latent structure for image synthesis and robotic imitation* in *Advances in neural information processing systems* (2006), 1233–1240.

66. Shridhar, K., Laumann, F. & Liwicki, M. Uncertainty Estimations by Softplus normalization in Bayesian Convolutional Neural Networks with Variational Inference. *arXiv preprint arXiv:1806.05978* (2018).

67. Sinha, S., Ebrahimi, S. & Darrell, T. *Variational adversarial active learning* in *Proceedings of the IEEE International Conference on Computer Vision* (2019), 5972–5981.

68. Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F. & Schmidhuber, J. *Compete to compute* in *Advances in neural information processing systems* (2013), 2310–2318.

69. Stallkamp, J., Schlipsing, M., Salmen, J. & Igel, C. *The German traffic sign recognition benchmark: a multi-class classification competition* in *Neural Networks (IJCNN), The 2011 International Joint Conference on* (2011), 1453–1460.

70. Tseran, H., Khan, M. E., Harada, T. & Bui, T. D. *Natural Variational Continual Learning* in *Continual Learning Workshop@ NeurIPS* **2** (2018).

71. Tzeng, E., Hoffman, J., Saenko, K. & Darrell, T. *Adversarial discriminative domain adaptation* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), 7167–7176.

72. Van Der Maaten, L. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research* **15,** 3221–3245 (2014).

73. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., *et al. Matching networks for one shot learning* in *Advances in neural information processing systems* (2016), 3630–3638.

74. Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* **11,** 37–57 (1985).

75. Xiao, H., Rasul, K. & Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, The MIT License (MIT) Copyright © 2017 Zalando SE. *https://tech.zalando.com, arXiv preprint arXiv:1708.07747* (2017).

76. Xu, C., Tao, D. & Xu, C. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634* (2013).

77. Yoon, J., Yang, E., Lee, J. & Hwang, S. J. *Lifelong Learning with Dynamically Expandable Networks* in *International Conference on Learning Representations* (2018).

78. Zenke, F., Poole, B. & Ganguli, S. *Continual Learning Through Synaptic Intelligence* in *Proceedings of the 34th International Conference on Machine Learning* (eds Precup, D. & Teh, Y. W.) **70** (PMLR, 2017), 3987–3995.

79. Zhang, M., Wang, T., Lim, J. H. & Feng, J. Prototype Reminding for Continual Learning. *arXiv preprint arXiv:1905.09447* (2019).