12-13-2023

Hyper-Learning with Deep Artificial Neurons

Brendan Blake Camp
Georgia State University

Hyper-Learning with Deep Artificial Neurons

by

Blake Camp

Under the Direction of Rolando Estrada, Ph.D. & Raj Sunderraman, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2023

ABSTRACT

Two problems have plagued artificial neural networks since their birth in the mid-20th century. The first is a tendency to lose previously acquired knowledge when there is a large shift in the underlying data-distribution, a phenomenon provocatively known as catastrophic forgetting. The second is an inability to know-what-they-don't-know, resulting in excessively confident behavior, even in uncertain or novel conditions. This text provides an in-depth history of these obstacles, complete with formal problem definitions and literature reviews. Most importantly, the proposed solutions herein demonstrate that these challenges can be overcome with the right architectures and training objectives. As this text will show, a thorough investigation of these topics necessitated several distinct approaches. Each of which, when considered in isolation, offers evidence that these problems are likely temporary obstacles on the path to true human-level intelligence. Lastly, we present a new learning framework called Hyper-Learning, which might allow both of these problems to be mitigated by a single architecture when coupled with the right training algorithm.

INDEX WORDS: Continual Learning, Active Learning, Open-Set Recognition, Meta-Learning, Neural Networks, Deep Artificial Neurons, Hyper-Learning, Deep Learning

Hyper-Learning with Deep Artificial Neurons

by

Blake Camp

Committee Chair:	Raj Sunderraman
Committee:	Rolando Estrada
	Jonathan Ji
	Xiaojun Cao
	Gennady Cymbalyuk

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

December 2023

DEDICATION

I dedicate this work to my Mother, my Father, and my Brother. I wish for it to serve as a testament to the enduring love and unwavering guidance of a family that consistently steered me towards the right path, even in the face of adversity. Collectively, they imparted the timeless wisdom that the most valuable pursuit in life is to embody love and kindness, irrespective of life's challenges, for this is the key to leaving behind a lasting legacy that outlives one's own achievements and existence.

ACKNOWLEDGMENTS

I attribute a significant portion of my Ph.D. journey's success to my advisor, Dr. Rolando Estrada. He consistently played the roles of mentor and compassionate human being, transforming the Ph.D. experience into a deeply gratifying journey. I must express my gratitude to Jaya Krishna Mandivarapu, not only as a close friend but also as a research partner. Over the last five years, he has provided me with excellent opportunities for collaboration and engaged in profound intellectual discussions spanning a wide spectrum of AI-related topics. A special acknowledgment goes to Dr. Raj Sunderraman, who provided invaluable assistance throughout my academic journey. His initial offer to join the CDC Research lab as a research assistant during my masters laid the foundation for my current accomplishments. This experience was truly enriching, and I acquired a wealth of knowledge. I'd also like to extend my appreciation to the Department of Computer Science for granting me the opportunity to pursue my Ph.D. Lastly, I am deeply honored to collaborate with my outstanding colleagues, who have enriched my academic journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A3C: Async Advantage Actor-Critic

ADAM: Adaptive Moment Estimation optimizer(An extension of SGD)

AE: Auto-Encoder

AGI: Artificial General Intelligence

AI: Artificial Intelligence

AL: Active Learning

ANN: Artificial Neural Network

CAE: Contractive Auto-Encoder

CF: Catastrophic Forgetting

CIL: Class Incremental Learning

CL: Continual Learning

DALBT: Deep Active Learning with Barlow Twins

DAN: Deep Artificial Neuron

DBAL: Deep Bayesian Active Learning

DNN: Deep Neural Network

ER: Experience Replay

EVT: Extreme Value Theory

EWC: Elastic Weight Consolidation

FF (a): Farthest-First (traversal principle)

FF (b): Feed-Forward

GAN: Generative Adversarial Network

GNN: Graph Neural Network

GRU: Gated-Recurrent Unit

IID: Independent and Identically Distributed

MAML: Model Agnostic Meta Learning

MSE: Mean Squared Error

OOD: Out-of-Distribution

OSR: Open-Set Recognition

RL: Reinforcement Learning

SGD: Stochastic Gradient Descent

SSL: Self-Supervised Learning

SVM: Support Vector Machine

TN: Task Network

VAAL: Variational Adversarial Active Learning

VECs: Vectorized Synapses

VNN: Variational Neural Network

WGD: Warped Gradient Descent

CHAPTER 1
Introduction

Lifelong or Continual Learning (CL) is one of the most formidable challenges in machine learning, and it remains a significant hurdle in the quest for artificial general intelligence (AGI). Within this intricate algorithmic landscape, a single system must constantly acquire the capability to tackle novel tasks while preserving its existing knowledge—a quest fraught with the specter of forgetting. These tasks can range from diverse data types - like images and text - to distinct ways of processing the same data, such as classification and segmentation. The relentless demand for continuous adaptation blurs the boundaries between training and inference, obliging a system to perpetually adjust its parameters.

CL is particularly challenging for deep neural networks because they are trained end-to-end. In standard deep learning we tune all of the network's parameters based on training data, usually via Backpropagation. While this paradigm has proven highly successful for individual tasks, it is not suitable for continual learning because it overwrites existing weights, a phenomenon evocatively dubbed catastrophic forgetting. For example, if we first train a network on task A, and then on a subsequent task B, the latter training will modify the weights learned for A, thus likely reducing the network's performance on this task. There are several approaches that can achieve some degree of continual learning in deep networks. However, existing methods suffer from various limitations. As this work will show, our collective understanding of the true challenges inherent to continual learning have evolved over time, as indicated by the various problem formulations and underlying objectives in

each forthcoming chapter.

Parallel to the complexities of CL, we encounter an entwined challenge, one that has echoed through the annals of artificial intelligence research for decades: Active Learning (AL). While supervised deep learning has achieved remarkable feats across diverse domains, the chasm between our capacity to amass data and our ability to label it remains glaringly evident [Lecun et al. 1998]. This discrepancy is most pronounced in domains requiring specialized expertise, like the world of medical imaging, where the need for ground truth annotations outpaces the availability of domain experts. Furthermore, considerations like time constraints, financial limitations, and environmental impact weigh heavily on the decisions inherent to training data acquisition and annotation.

The history of AL is rooted in this conundrum. It dates back to an era where computational resources were scarce, data labeling was labor-intensive, and the quest to harness the power of machine learning was in its infancy. As we embarked on the journey of automating the selection of data samples for annotation, a pivotal question emerged: How can we maximize the efficiency of data acquisition while minimizing the human labeling effort? This question catalyzed the birth of active learning, where the selection of informative samples for labeling became a core pursuit. The fundamental premise underlying AL is that not all data samples are equally informative. Rather, there exists a select subset—often relatively small—that can furnish the majority of information needed for effective model training. This subset represents the crux of AL's historical mission: to optimize model performance within predefined constraints.

In the next sections, we delve deeper into the evolving landscape of active learning, illuminating its transformative impact on artificial intelligence research and revealing its intricate relationship with the continual learning paradigm. Through our exploration, we aim to capture the essence of this historical continuum, highlighting its enduring relevance in our quest for AGI.

Chapter 2 introduces Self-Net, a groundbreaking framework that confronts the continual learning challenge head-on. Leveraging autoencoders, Self-Net learns low-dimensional representations of network weights, facilitating the seamless integration of new tasks with minimal retraining and without the need for extensive data storage. This innovation opens doors to efficient and adaptive continual learning, boasting superior storage performance across diverse learning paradigms.

Chapter 3 begins our exploration into the domain of active learning, redefining it as an open-set recognition challenge. Harnessing the power of variational neural networks, this paper quantifies uncertainty and selectively identifies informative samples for labeling, effectively addressing the challenge of resource-efficient data acquisition. Through a probabilistic approach, it attains state-of-the-art results across a range of datasets, underscoring its capacity to manage uncertainty for informed active learning.

In Chapter 4, the fusion of active learning with self-supervised learning results in Deep Active Learning with Barlow Twins (DALBT)—a novel methodology that leverages self-supervised learning alongside a simple classifier. This innovation offers a scalable and efficient approach, amd the introduction of Weibull sampling enhances the sample selection's diversity

and utility. DALBT's versatility is substantiated by its exceptional performance across datasets like MNIST, Fashion-MNIST, and CIFAR-10.

Chapter 5 ventures into uncharted territory with the introduction of Deep Artificial Neurons (DANs). Inspired by the intricate dynamics of biological neurons, DANs operate as vector-valued filters, mitigating catastrophic forgetting and enabling seamless continual learning. This non-linear, vector-valued filtering demonstrates the potential of complex neural dynamics in the pursuit of lifelong learning.

Our odyssey culminates in Chapter 6, where we delve into the realm of Hyper-Learning—a paradigm that redefines the role of individual neurons, challenging traditional notions of neural simplicity. DANs embedded within larger networks serve as the nucleus of Hyper-Learning, self-supervising synaptic updates to address challenges like catastrophic forgetting and suboptimal generalization. Through population-based self-supervised meta-learning, Hyper-Learning emerges as a transformative paradigm poised to unlock the boundless potential of deep nueral networks.

This narrative of innovation and exploration weaves together these milestones, revealing a journey marked by transformative insights as we strive to unlock the limitless horizons of artificial intelligence.

1.1 Special Acknowledgement of Collaboration

On a final note, I would like to express my heartfelt gratitude to my research partner, Jaya Krishna Mandivarapu, for his unwavering friendship and invaluable research collaboration

throughout our journey as PhD students. The cornerstone of our collaboration was the groundbreaking paper "Self-Net: Lifelong Learning via Continual Self-Modeling", which we co-authored as the first of our joint endeavors. In this project, Jaya and I stood as co-first authors, reflecting the harmonious collaboration that defined our academic journey. Jaya's contributions to the dual fields of Continual Learning and Active Learning have been exceptional, further demonstrated by his role as the first author on two additional pivotal papers: "Deep Active Learning via Open-Set Recognition" and "Deep Active Learning with Barlow Twins." In these works, I take immense pride in having played a significant, yet assisting role in the conceptual design of ideas, the development of targeted code, as well as assistance in writing the papers themselves, all of which enabled the successful realization of these innovative projects. Jaya's dedication and our collaborative efforts have significantly enriched my academic experience, and for that, I am truly grateful.

CHAPTER 2

Self-Net: Lifelong Learning via Continual Self-Modeling

2.1 Problem Formulation

Continual learning is not a single problem, but a family of related problems, each of which imposes a different set of constraints on the learning process (e.g., fixed architecture, no access to prior training data, etc.). Here, we consider the setting in which (1) the system learns one new task at a time, (2) each task can be solved independently of other tasks, (3) tasks have labels (i.e., the system knows which task to solve at any point), and (4) the system has no access to old training data. In particular, our problem differs from settings in which a single task grows more difficult over time [e.g., class-incremental learning (CIL)].

More concretely, each task T_i is specified by a training set, $D_i = \{X_i, Y_i\}$, consisting of n_i different $\{x, y\}$ training pairs. The system is sequentially trained on each D_i dataset, using either a supervised or reinforcement learning paradigm, as applicable. That is, the system is first exposed to D_1 (and thus must learn T_1), then D_2, D_3, up to D_k, where k is the total number of tasks encountered during its lifetime. Note that, in this paradigm, datasets are not required to be disjoint, i.e., any two datasets D_i and D_j many contain some common $\{x, y\}$ pairs.

Critically, the system is trained on each D_i only once during its lifetime. The system is not allowed to store any exemplars from previous tasks or revisit old data when training on new tasks. We do, however, allow multiple passes over the data when first learning the task, as is standard in machine learning. We also assume that task labels are known; inferring the

desired task from the input data is important but is outside the scope of this work.

There are two common types of solutions for this CL problem. Regularization methods estimate a single set of parameters θ^* for all tasks, while growth-based approaches learn (and store) a new set of weights θ_i for each new task. The former uses constant storage (w.r.t to the number of tasks) but has bad performance, while the latter achieves good performance but is asymptotically equivalent to storing independent networks. Below, we detail our proposed approach, which has nearly the same performance as growth-based methods, but uses significantly less storage.

2.2 Methodology

Figure 2.1 provides a high-level overview of our proposed approach which we call **Self-Net**. Self-Net is instantiated as a single Autoencoder (AE) and is charged with modeling the independent Task networks (TN) that are used to learn each subsequent task encountered by the system. At any given time step after solving the first task the proposed system utilizes an m-dimensional Buffer for storing newly learned tasks, an $O(n)$ lifelong autoencoder (AE) for storing older tasks, and single s-dimensional latent vector for each task. This s-dimensional latent vector for each task is saved, where $s << n$ which means that the size of the latent vector is far smaller than the size of the task network which was used to solve the task. Assuming that c and m are constants, our space complexity is $O(n + ks)$, where k is the number of learned tasks. In particular, the proposed approach achieves asymptotic space savings compared to storing kn independent networks if s is sub-linear w.r.t. n, (i.e., $s = \omega(n)$

in asymptotic notation).

One of the main advantages of the proposed approach is that each task network (TN) is just an independent and standard neural network, which can learn regression, classification, or reinforcement learning (RL) tasks (or some combination of the three as shown in the experimental section). For ease of discussion, we will focus on the case where there is a single TN and the Buffer can hold only one network; this can be easily extended to multiple networks. The AE is made up of an *encoder* that compresses an input vector into a lower-dimensional, latent vector e, and a *decoder* that maps e back to the higher-dimensional space. Our system can produce high-fidelity recollections of the learned weights, despite this intermediate compression. In our experiments, we used a contractive autoencoder (CAE) due to its ability to quickly incorporate new values into its latent space.

In CL, we must learn k different tasks sequentially. To learn these tasks independently, one would need to train and save k networks, with $O(n)$ parameters each, for a total of $O(kn)$ space. In contrast, we propose using our AE to encode each of these k networks as an s-dimensional latent vector, with $s << n$. Thus, our method uses only $O(n+ks)$ space, where the $O(n)$ term accounts for the TNs and the fixed-size Buffer. Despite this compression, our experiments show that we can obtain a high-quality approximation of previously learned weights, even when the number of tasks exceeds the number of parameters in the AE. Below, we first describe how to encode a single task-network before discussing how to encode multiple tasks in continual fashion.

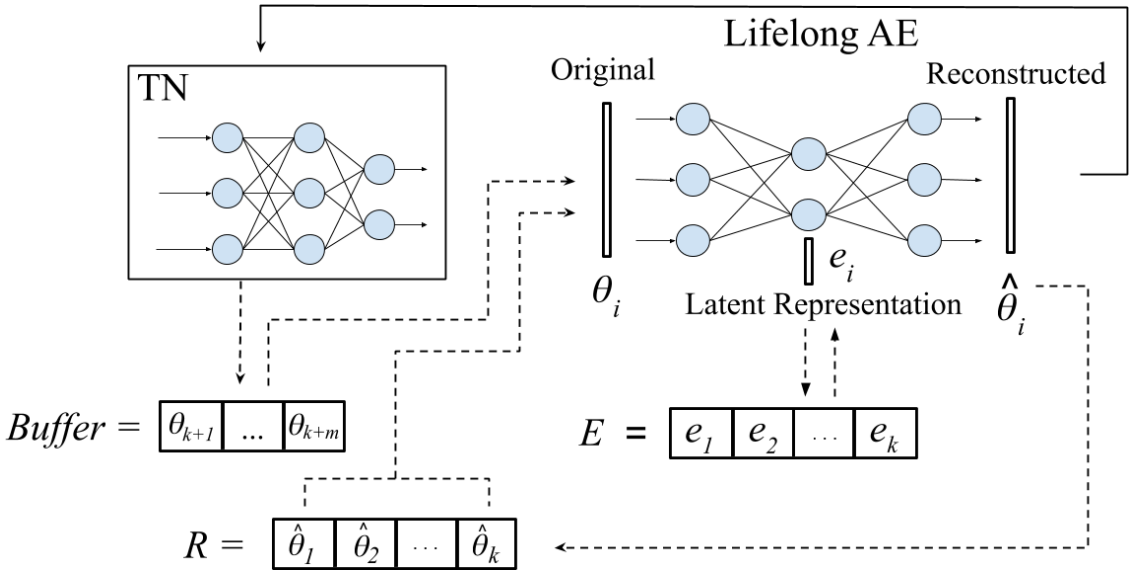


Figure 2.1 **Framework overview:** Our proposed system has a set of reusable *task-specific networks* (TN), a *Buffer* for storing the latest m tasks, and a lifelong, *auto-encoder* (AE) for long-term storage. Given new tasks $\{t_{k+1}, ..., t_{k+m}\}$, where k is the number of tasks previously encountered, we first train m task-networks independently to learn $\{\theta_{k+1}, ..., \theta_{k+m}\}$ optimal parameters for these tasks. These networks are temporarily stored in the Buffer. When the Buffer fills up, we incorporate the new networks into our long-term representation by retraining the AE on both its approximations of previously learned networks and the new batch of networks. When an old network is needed (e.g., when a task is revisited), we reconstruct its weights and load them onto the corresponding TN (solid arrow). Even when the latent representation e_i is asymptotically smaller than θ_i, the reconstructed network closely approximates the performance of the original.

2.2.1 Single-network encoding

Without loss of generality, we will now explain the protocol for encoding a single skill (task-network), and in the coming sections we will show that it can be easily applied to multiple skills in continual fashion. Let t be a task (e.g., classifying digits) and let $f(\theta_t)$ be the network used to solve that task. Once solved, the task-network $f(\theta_t)$ can be flattened into an $O(n)$-dimensional vector of parameters. That is, using a task-network with parameters θ, we can achieve performance p on t (e.g., a classification accuracy of 95%). Now, let $\hat{\theta}$

be the approximate reconstruction of θ by our autoencoder and let $\hat{p}$ be the performance that we obtain by using these reconstructed weights for task t. Our goal is to minimize any performance loss w.r.t. the original weights. If the performance of the reconstructed weights is acceptable, then we can simply store the $O(s)$ latent vector e, instead of the $O(n)$ original vector θ.

If we had access to the test data for t, we could assess this difference in performance directly and train our AE until we achieved an acceptable margin ϵ:

$$p - \hat{p} \leq \epsilon. \tag{2.1}$$

For example, for a classification task we could stop training our AE if the drop in accuracy is less than 1%.

In a continual learning setting, though, the above scheme requires storing validation data for each old task. Instead, we measure a distance between the original and reconstructed weights and stop training when we achieve a suitably close approximation. Empirically, we determined that the cosine similarity,

$$\cos(\theta, \hat{\theta}) = \frac{\theta \cdot \hat{\theta}}{\|\theta\|\|\hat{\theta}\|} = \frac{\sum_{i=1}^{n} \theta_i \hat{\theta}_i}{\sqrt{\sum_{i=1}^{n} \theta_i^2} \sqrt{\sum_{i=1}^{n} \hat{\theta}_i^2}}, \tag{2.2}$$

is an excellent proxy for a network's performance. Unlike the mean-squared error, this distance metric is scale-invariant, so it is equally suitable for weights of different scales, which may be the case for separate networks trained on distinct tasks. As detailed in Section 2.4,

cosine similarity close to 0.99 yielded excellent performance for a wide variety of tasks and architectures.

2.2.2 Continual encoding

In this section, we will now detail how to use our Self-Net to encode a sequence of trained networks in a continual fashion, thereby obviating the need to continually retrain on all historical data. Let m be the size of the Buffer, and let k be the number of tasks which have been previously encountered. As noted above, we train each of these m task-networks using conventional backpropagation, one per task. Now, assume that our AE has already learned to encode the first k task-networks. We will now show how to encode the most recent batch of m task-networks corresponding to tasks $\{t_{k+1}, ..., t_{k+m}\}$ into compressed representations $\{e_{k+1}, ..., e_{k+m}\}$ while still remembering all previously trained networks.

Let E be the set of latent vectors for the first k networks. In order to integrate m new networks $\{\theta_{k+1}, ..., \theta_{k+m}\}$ into the latent space, we first recollect all previously trained networks by feeding each $e \in E$ as input to the decoder of the AE. We thus generate a set R of recollections, or approximations, of the original networks (see Fig. 2.1). We then append each θ_i in the Buffer to R and retrain the AE on all $k + m$ networks until it can reconstruct them, i.e., until the average of their respective cosine similarities is above the predefined threshold. Algorithm 1 summarizes our CL strategy.

As our experiments show, our compressed representations achieve excellent performance compared to the original parameters. Since each $\hat{\theta} \in R$ is simply a vector of network

Algorithm 1 Lifelong Learning via Continual Self-Modeling

```
Let T be the set of all Tasks encountered during the lifetime of the system
Let m be the size of the Buffer
Set E = []
Initialize AE
Set cosine_threshold
for idx,curr_task in enumerate(T) do
    if Buffer is not full then
        Initialize the Task Network (TN)
        Train the TN for curr_task until optimized
        Buffer.append(TN)
        if Buffer is full then
            R = []
            for encoded-network in E do
                r = AE.Decoder(encoded-network)
                R.append(r)
            for network in Buffer do
                flat_network = extract and flatten parameters from network
                R.append(flat_network)
            average_cosine_similarity = 0.0
            E = []
            while average_cosine_similarity < cosine_threshold do
                for r_idx,r ∈ enumerate(R) do
                    calculate AE_loss using Equation 4
                    back-propagate AE
                    update average_cosine_similarity using
                    cos(r,AE(r))
                    E[r_idx] = AE.Encoder(r)
            empty Buffer
```

parameters, it can easily be loaded back onto a task-network with the correct architecture. We can thus discard the original networks and store k networks using only $O(n+ks)$ space. In addition, our framework can encode many different types and sizes of networks in a continual fashion. In particular, we can encode a network of arbitrary size q using a constant-size AE

(that takes inputs of size n) by splitting the input network into r subvectors[1], such that $(n = q/r)$. As we verify in Section 2.4, we can effectively reconstruct a large network from its subvectors and still achieve a suitable performance threshold.

As Fig. 2.2 illustrates, we empirically found a strong correlation between a reconstructed network's performance and its cosine similarity w.r.t. to the original network. Intuitively, this implies that vectors of network parameters that have a cosine similarity approaching 1 will exhibit near-identical performance on the underlying task. Thus, the cosine similarity can be used as a terminating condition during retraining of the AE. In practice, we found a threshold of .997 to be sufficient for most experiments.

2.2.3 Autoencoder details

Our proposed framework is agnostic to the choice of autoencoder. However, in our experiments we used contractive autoencoders (CAE) because we empirically found them to be more robust than other types of AEs, including variational autoencoders. CAEs are identical to standard AEs, except that their loss function penalizes changes to the latent vector's values:

$$CAE_{loss}(\theta) = \cos(\theta, \hat{\theta}) + \lambda \|J_f(\theta)\|_F^2. \tag{2.3}$$

[1]We pad with zeros whenever q and n are not multiples of each other.

The first term is the cosine similarity discussed above (see Eq. 2.2), while the regularization term is given by the Frobenius norm of the Jacobian w.r.t to each training input x_i:

$$\|J_f(x)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(\theta)}{\partial x_i} \right)^2. \tag{2.4}$$

where $h_j(\theta)$ are the parameters for the j-th hidden unit. In our experiments, we used a value of 0.0001 for lambda.

2.3 Task network fine-tuning

As an additional optimization, one can improve the speed with which the AE learns a new task by encouraging the parameters of new task-networks to be as similar as possible to previously learned ones. This can be accomplished by fine-tuning all networks from a common source and penalizing large deviations from this initial configuration with a regularization term. Note that training new task networks in this manner differs from standard regularization methods (e.g., EWC) because the weights learned for older tasks are not modified (and hence their performance does not degrade).

Formally, let θ^* be the source parameters, ideally optimized for some highly-related task. Without loss of generality, we can define the loss function of task-network θ_i for task t_i as:

$$TaskNetLoss_i = TaskLoss + \lambda MSE(\theta^*, \theta_i) \tag{2.5}$$

where λ is a regularization coefficient that determines the importance of remaining close to

the source parameters vs. optimizing for the current task. By encouraging the parameters for all task-networks to remain close to one another, we make it easier for the AE to learn a low-dimensional representation of the original space. We employ this scheme for the experiments section with $\lambda = 0.001$.

2.4 Results

We carried out a range of CL experiments on a variety of datasets in both supervised and reinforcement-learning (RL) settings. First, we performed a robustness analysis in order to empirically establish how precise an approximation of a network must be in order to retain comparable performance on a task. Then, we analyzed our system's ability to encode a very large number of tasks, thus validating that the AE does simply memorize the TNs. We then evaluated the performance of our approach on the following CL datasets: Permuted MNIST [Kirkpatrick et al. 2017], Split MNIST [Nguyen et al. 2018], Split CIFAR-10 [Zenke et al. 2017], Split CIFAR-100 [Zenke et al. 2017], and successive Atari games [Mnih et al. 2013] (we describe each dataset below). Finally, we also analyzed our system's performance when using **(2)** different sizes of AEs, and **(3)** different TN architectures.

2.4.1 Robustness Analysis

In our initial experiments, we added different levels of i.i.d, zero-mean Gaussian noise to the weights of a trained network. Our goal was twofold: **(1)** to verify that approximate weights can differ from their original values while still retaining good performance and **(2)** to establish a threshold at which to stop training our AE. Since we assume no access to data

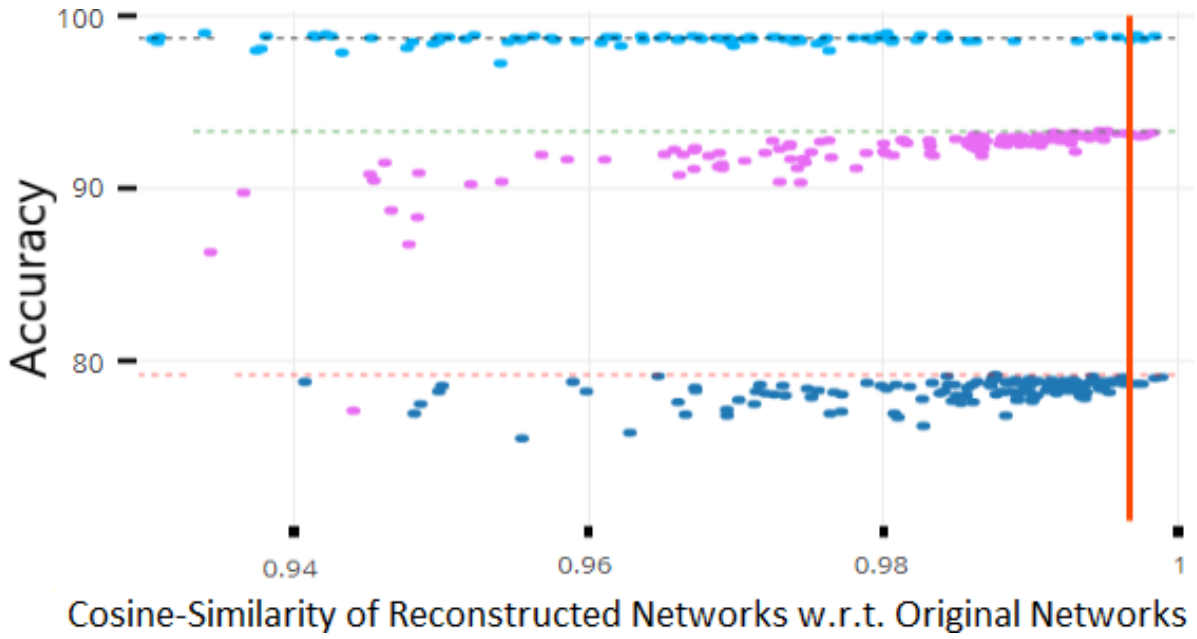


Figure 2.2 **Robustness analysis of network performance as a function of cosine similarity:** Each dot represents the accuracy of a reconstructed network and the dotted lines are the baseline performances of the original networks. The above values for three datasets (Permuted MNIST (in pink), MNIST (in cyan), and CIFAR-10 (in blue), show that cosine similarity values above 0.997 guarantee nearly optimal performance for these datasets.

from previously learned tasks, we need a way to estimate the performance of a reconstructed network without testing on a validation set.

Figure 2.2 shows performance as a function of deviations from the original parameters as measured by cosine similarity, for three datasets (described below). Under this metric, there is a clear correlation between the amount of parameter dissimilarity and the probability of a decrease in performance. The red line indicates a cosine similarity of 0.997. Weights above this value had nearly identical performance to the original values. Thus, unless otherwise noted, we used this threshold as a terminating condition in our subsequent experiments.

2.4.2 Performance and Scalability

In the next set of experiments, we verified that our method retains excellent performance even when the number of TN parameters exceeds the number of parameters in the AE. In other words, here we confirmed that our AE is *compressing previously learned weights*, not simply memorizing them. More generally, there is a trade-off in CL between storage and performance. Using different networks for k tasks yields optimal performance but uses $O(kn)$ space, while regularized methods such as Online EWC [Huszár 2018] only require $O(n)$ space but suffer a steep drop in performance as the number of tasks grows. For any method, we can quantify performance as a *compression factor*, i.e., the number of additional parameters it stores per task; in our case, our compression factor is k/s because we store an s-dimensional vector per task.

Here, our experimental paradigm was as described in Sec. 2.2.2: we first trained the TN on m tasks independently, storing each set of learned weights in the Buffer. Once the Buffer became full, we trained the AE to encode these weights into its latent space, only storing the latent vectors after training. We then continued to train the TN on new batches of m tasks (saving the new weights to the Buffer). Every time the Buffer became full, we trained the AE on *all tasks*, using the stored latent vectors and the new m weights. After the initial batch, we fine-tuned all networks from the mean of the initial set of m networks and penalized deviations from this source vector (using $\lambda = 0.001$), as described in Section 2.2.

For these experiments, we used the Split MNIST dataset [Nguyen et al. 2018], which consists of different binary subsets of the MNIST dataset [LeCun et al. 1998], drawn ran-

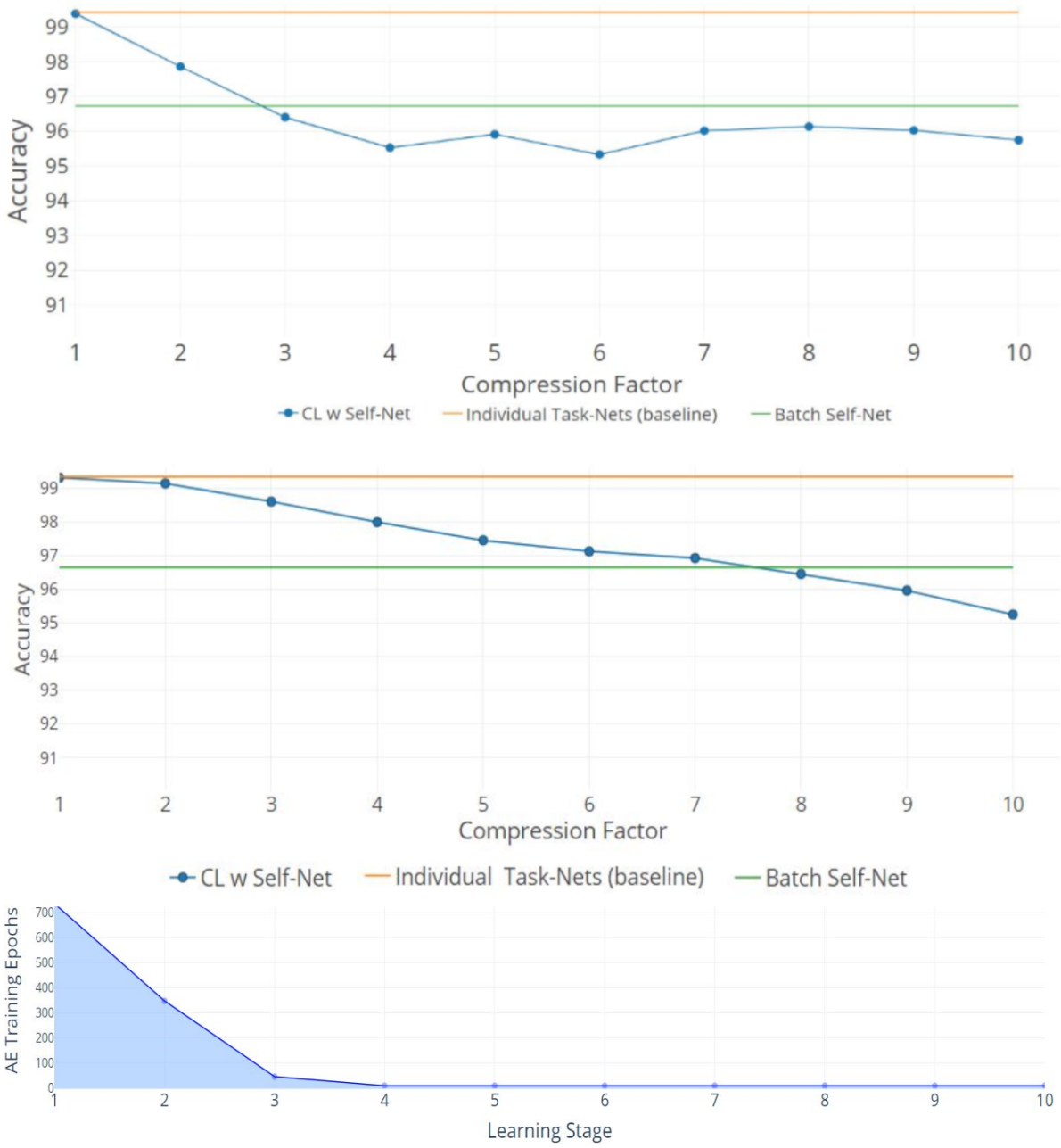


Figure 2.3 **10X Compression for Split-MNIST:** Orange lines denote the average accuracy achieved by individual networks, one per task. Green lines denote the average accuracy when training the AE to encode all networks as a single batch. Blue lines indicate the average accuracy obtained by Self-Net at each CL Stage. **Top:** 50 tasks with latent vectors of size 5 and a Buffer of size 5. **Middle:** 100 tasks with latent vectors of size 10 and Buffer of size 10. The x-axis (top and middle) denotes the compression factor achieved at each learning stage. **Bottom:** the training epochs required by the 5-dimensional AE to incorporate new networks decreases rapidly over time.

domly. In other words, tasks were defined by tuples comprised of the positive and negative digit class(es), e.g., ([pos={1}, neg={6,7,8,9}], [pos={6}, neg={1,2,3,4}], etc.). Here, the training and test sets consisted of approximately 40% positive examples and 60% negative examples. For this experiment, we trained a deep convolutional task network with 2 convolution layers (kernels of size 5x5 and stride 1x1), 1 hidden layer (320x50), and 1 output layer (50x10)—21,840 parameters in total. Our task network used ReLU activation units. Our AE, on the other hand, had one fully connected hidden layer with either 5 or 10 units. We used a Buffer of the same size as the latent vector, i.e., either 5 or 10. These values were chosen so that each new batch of networks yielded an integer compression factor, e.g., encoding 15 networks with a latent vector of size 5 gives 3X compression ($k/s = 3$). We used decreasing thresholds to stop training our AE: 0.9996 for the initial batch, 0.987 for the second batch, and 0.986 for subsequent batches.

The top two plots of Fig. 2.3 show the mean performance for up to 50 and 100 Split-MNIST tasks, given latent vectors of size 5 and 10, resp. All figures show the average accuracy across all tasks learned up to that point. For comparison, we also plotted the original networks' performance and the performance of the reconstructions when the AE learned all the tasks in a single batch (green and orange lines, resp.). The line with dots represents the CL system; each dot indicates the point where the AE had to encode a new set of m networks. For 10X compression, the Self-Net with a latent vector of size 5 retained ∼95.7% average performance across 50 Split-MNIST tasks, while the Self-Net with 10-dimensional latent vectors retained ∼95.2% across 100 tasks. This represents a relative

change of only ∼3.3% compared to the original performance of ∼99%. In other words, our approach is able to compress 21,840 parameters into 5 or 10 values with little performance loss, even when trained in a continual fashion. In contrast, existing methods dropped to ∼50% performance after learning only 10 tasks on this dataset (see Fig. 2.4 below). Finally, we note that by initializing each new network from the mean of the initial batch, our AE was able to incorporate subsequent networks with very little additional training (see stages 4-10 in bottom image of Fig. 2.3).

2.4.3 Permuted MNIST

In the next set of experiments, we compared our approach to state-of-the-art methods across multiple datasets. First, we trained convolutional feed-forward neural networks with 21,840 parameters on successive tasks, each defined by distinct permutations of the MNIST dataset [LeCun et al. 1998], for 10-digit classification. We used networks with 2 convolution layers (kernels of size 5x5, and stride 1x1), 1 hidden layer (320x50), and 1 output layer (50x10). Our AE had three, fully connected layers with 21,840, 2000, and 20 parameters, resp. Thus, our latent vectors were of size 20. For this experiment, we used a Buffer of size 1. Each task network was encoded by our AE in sequential fashion, and the accuracies of all reconstructed networks were examined at the end of each learning stage (i.e., after learning a new task). Figure 2.4 (top) shows the mean performance after each stage for all tasks learned up to that point. Our technique almost perfectly matched the performances achieved by independently trained networks, and it dramatically outperformed other state-of-the-art approaches including EWC [Kirkpatrick et al. 2017], Online EWC (the correction to EWC proposed in

[Huszár 2018]), and Progress & Compress [Schwarz et al. 2018]. As a baseline, we also show the results for SGD (no regularization), L2-based regularization in which we compare new weights to all the previous weights, and Online L2, which only measures deviations from the weights learned in the previous iteration. Our technique remembers old tasks without inhibiting new learning.

2.4.4 Split MNIST

We the compared our method to the same set of prior approaches on Split MNIST (described above). Our task-networks, CAE, and Buffer size remained the same as what was used for our Permuted MNIST experiments (except that the outputs of the task-networks were binary, instead of 10 classes). In this domain, too, our technique dramatically outperformed competing approaches, as seen in Figure 2.4 (middle).

2.4.5 Split CIFAR-10

We then verified that our proposed approach could reconstruct larger, more sophisticated networks. Similar to the Split MNIST experiments above, we divided the CIFAR-10 dataset [Krizhevsky 2009] into multiple training and test sets, yielding 10 binary classification tasks (one per class). We then trained a task-specific network on each class. Here, we used TNs having an architecture which consisted of 2 convolutional layers, followed by 3 fully connected hidden layers, and a final layer having 2 output units. In all, these task networks consisted of more than 60K parameters. Again, for this experiment we used a Buffer of size 1. Our AE had three, fully connected layers with 20442, 1000, and 50 parameters, resp. As

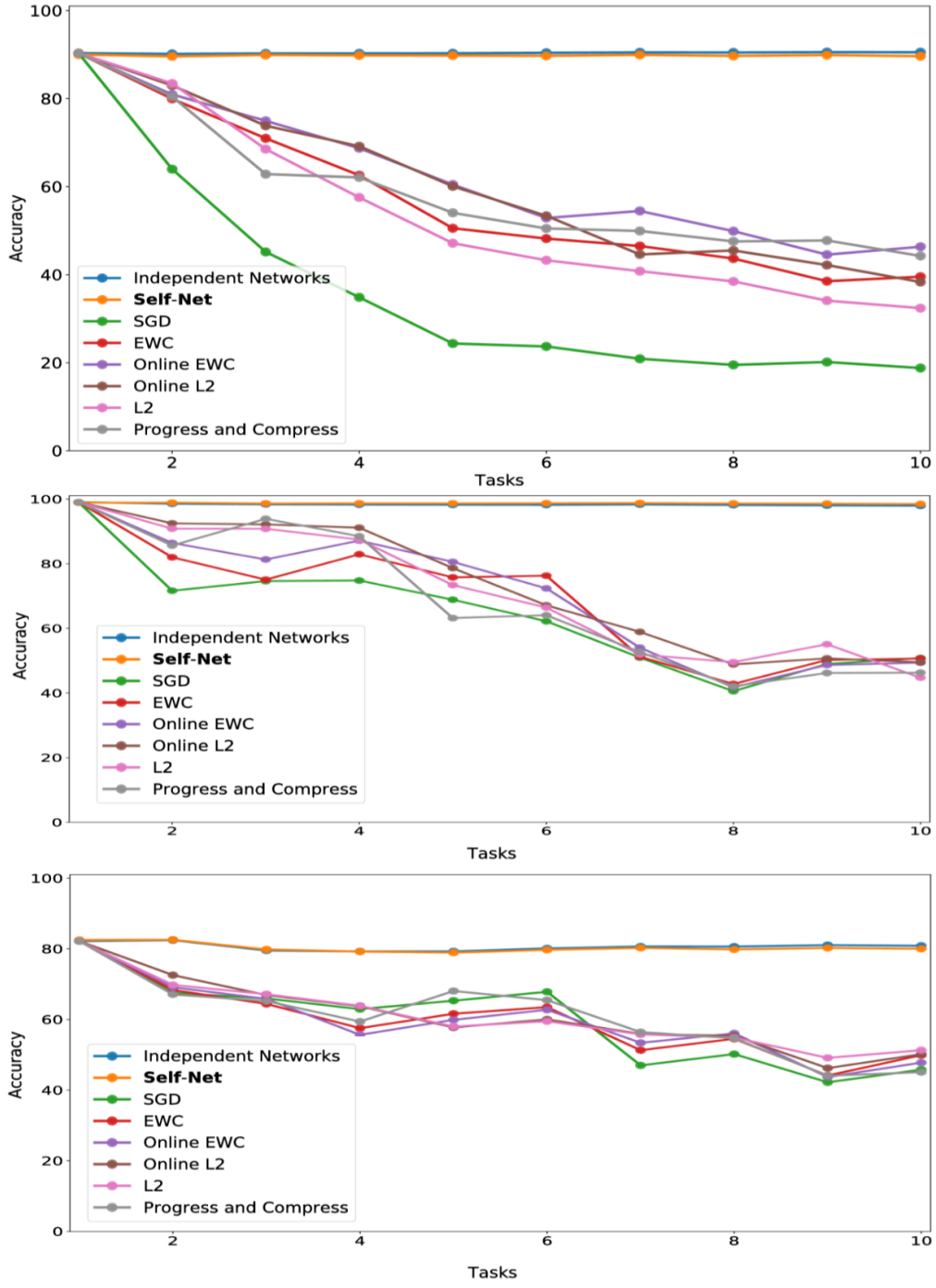


Figure 2.4 CL performance comparisons with average test set accuracy on all observed tasks at each stage for **(top)** Permuted MNIST, **(middle)** Split MNIST, and **(bottom)** Split CIFAR-10.

described in Sec. 2.2, we split the 60K networks into three subvectors to encode them with our autoencoder; by splitting a larger input vector into smaller subvectors, we can encode networks of arbitrary sizes. The individual task-networks achieved accuracies ranging from 78% to 84%, and a mean accuracy of approximate 81%. Importantly, we encoded these larger networks using almost the same AE architecture as the one used in the MNIST experiments. As seen in Figure 2.4 (bottom), the accuracies of the reconstructed CIFAR networks also nearly matched the performances of their original counterparts, while also outperforming all other techniques.

2.4.6 Split CIFAR-100

We applied a similar approach for the CIFAR-100 dataset [Krizhevsky 2009]. That is, we split the dataset into 10 distinct batches comprised of 10 classes of images each. We used the same task-network architecture and Buffer size as in our CIFAR-10 experiments, modified slightly to accommodate a 10-class classification objective. The trained networks achieved accuracies ranging from 46% to 49%. We then encoded these networks using the same AE architecture described in the previous experiments, again accounting for the input size discrepancy by splitting the task-networks into smaller subvectors. As seen in Figure 2.5, our technique almost perfectly matched the performances achieved by independently trained networks.

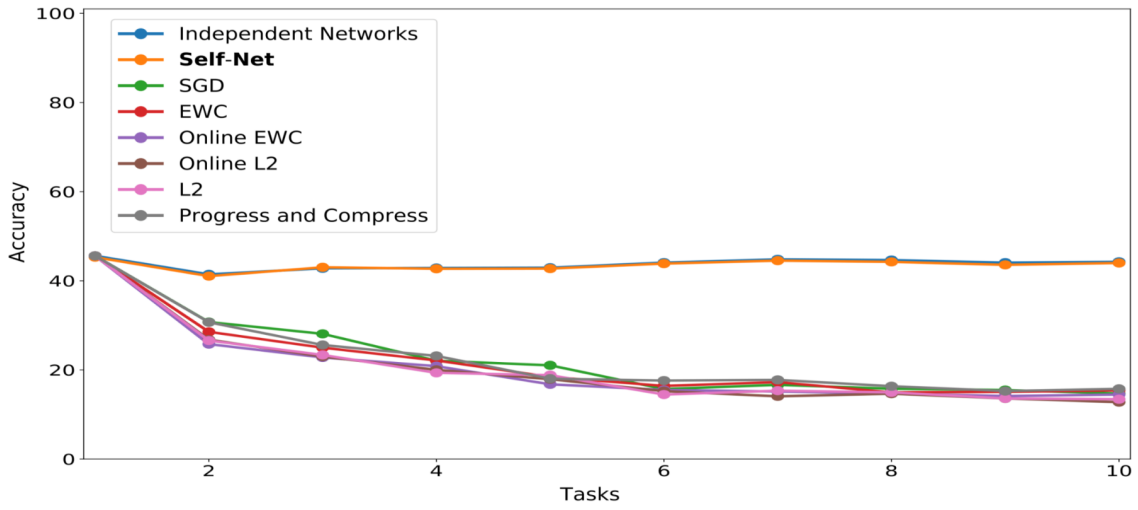


Figure 2.5 CL performance comparisons with average test set accuracy on all observed tasks at each stage for CIFAR-100.

2.4.7 Incremental Atari

To evaluate the CL performance of Self-Net in the challenging context of reinforcement learning, we used the code available at [Greydanus 2017] to implement a modified Async Advantage Actor-Critic (A3C) framework; this architecture, originally introduced in [Mnih et al. 2016], can learn successive Atari games while retaining good performance across all games. The model we used had 4 convolutional layers (kernals of size 3x3, and strides of size 2x2), a GRU layer (800x256), and two ouput layers: an Actor (256xNum_Actions), and Critic (256x1), resulting in a complex model architecture and over 800K parameters. Critically, this entire model can be flattened and encoded by the single AE in our Self-Net framework having three, fully connected layers with 76863, 2000, and 200 parameters, resp. For these experiments we also used a Buffer of size 1.

Similar to previous experiments, we trained our system on successive tasks, specifically

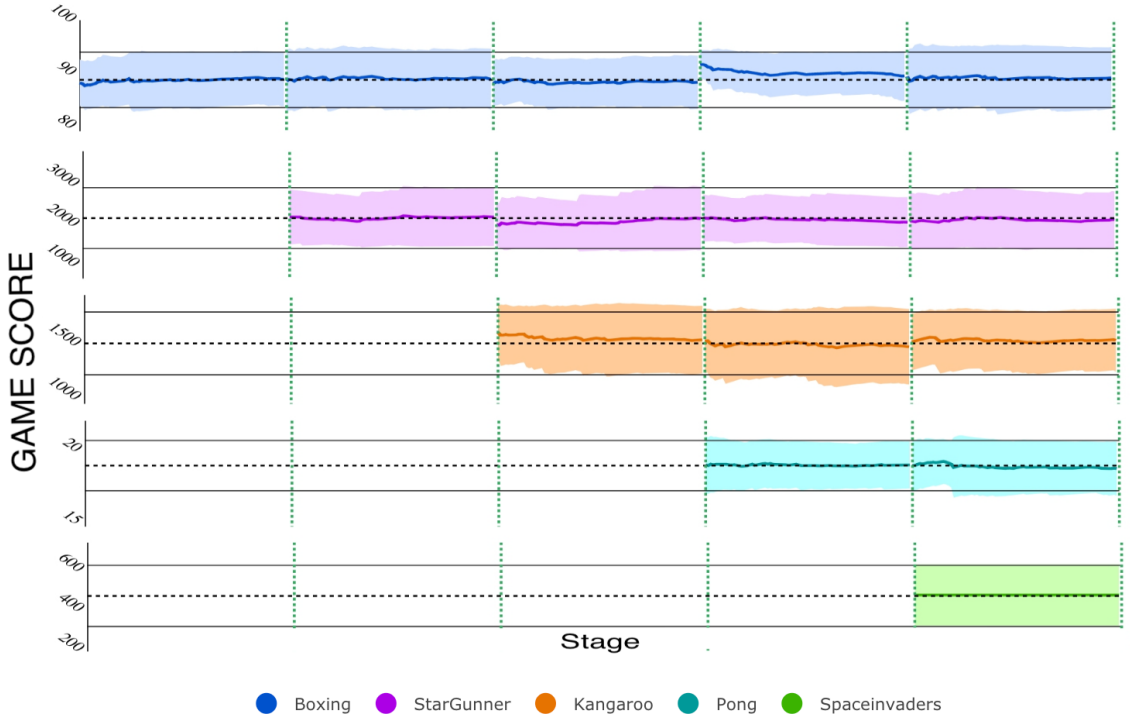


Figure 2.6 **CL on five Atari games with Self-Net:** To evaluate the reconstruction score at each stage, we ran the reconstructed networks for 80 full game episodes. The colored lines and bands represent the running mean and standard deviation in game score per episode. The cumulative mean score is nearly identical to the original TN at each stage.

the following Atari games: Boxing, Star Gunner, Kangaroo, Pong, and Space Invaders. Figure 2.6 shows the near-perfect retention of performance on each of the 5 games over the lifetime of the system. This was accomplished by training on each game only once, never revisiting the game for training purposes. The dashed, vertical lines demarcate the different stages of continual learning. That is, each stage indicates that a new network was trained for a new game, over 40M frames. Afterwards, the mean (dashed, horizontal black lines) and standard-deviation (solid, horizontal black lines) of the network's performance were computed by allowing it to play the game, unrestricted, for 80 episodes. After each stage,

the performances of all reconstructed networks were examined by re-playing each game with the appropriate reconstructed network. As Figure 2.6 shows, the cumulative means and SD's of the reconstructed networks closely mimic those achieved by their original counterparts.

2.4.8 Split Networks and Multiple Architectures

Finally, we verified that **(1)** a smaller AE can encode multiple network splits in substantially less time than a larger one can learn the entire network and **(2)** that the same AE can be used to encode trained networks of different sizes and architectures. Figure 2.7 (left) shows the respective training rates of an AE with 20,000 input units (blue line)—trained to reconstruct 3 sub-vectors of length 20,000—compared to that of a larger one, with 61,000 input units (yellow line), trained on a single 60K CIFAR-10 network. Clearly, using more inputs for a smaller AE enables us to more quickly encode larger networks. Finally, Figure 2.7 (right) shows that the same AE can simultaneously reconstruct 5 MNIST networks and 1 CIFAR network so that all networks approach their original accuracies.

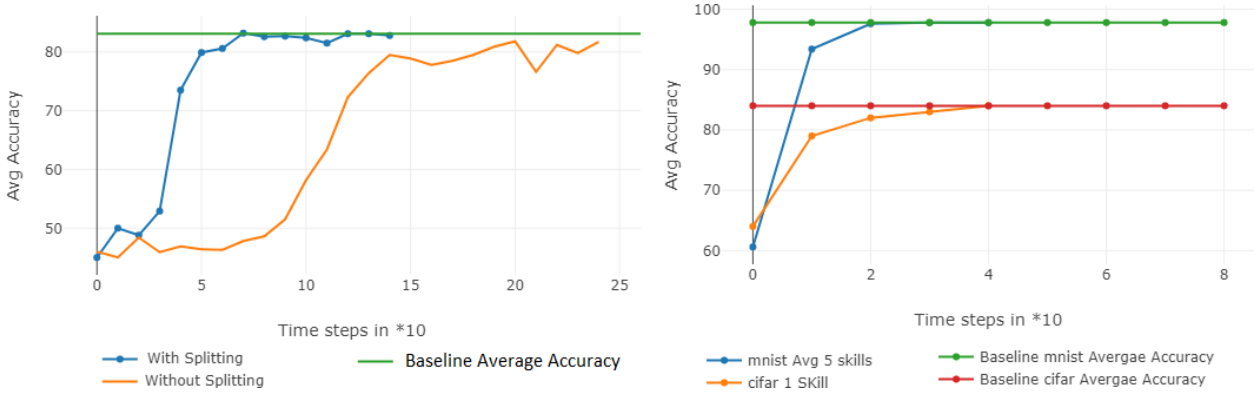


Figure 2.7 **Additional analyses: Left:** the AE training efficiency is improved when large networks are split into smaller subvectors. **Right:** a single AE can encode networks of different architectures and sizes.

2.5 Self-Net Summary

2.5.1 Initial Conclusions

In this paper, we introduced a scalable approach for multi-context continual learning that decouples how to learn a set of parameters from how to store them for future use. Our proposed framework uses state-of-the-art autoencoders to facilitate lifelong learning via continual self-modeling. Our empirical results confirm that our method can efficiently acquire and retain large numbers of tasks in continual fashion. In future work, we plan to further improve our autoencoder's capacity and explore how to use the latent space to extrapolate to new tasks using little or no training data. We also intend to compress the latent space even further (e.g., using only $\log(k)$ latent vectors for k tasks). Promising approaches include clustering the latent vectors into sets of related tasks or using sparse latent representations. Finally, we will also investigate how to infer the current task automatically.

2.5.2 Hindsight Observations & Future Work

Keen observers may notice some inherent flaws in the way that Self-Net operates. This is expected and welcomed. For instance, The method, as originally proposed, does not allow for automatic task inference, and therefore does not provide a protocol for deciding which task-network to retrieve in the event that there may be underlying task overlap between networks. There are other, more subtle oversights that would also need to be addressed in realistic deployment settings. For example, an astute reader may have notice that the Atari experiments do not actually employ any form of compression. This is because the size of the underlying Autoencoder is significantly larger than the aggregate total of all independently trained Atari Task-Networks. As a result, it should be intuitive that there is little-to-no drop in performance across any of the games after continual learning. It is this author's opinion that the reader should accept those experiments as a compelling proof of concept. Without additional, explicit evidence to the contrary, we would expect the observed performance trends to hold, once compression starts to occur as the number of games and task-networks grows.

The true measure of a paper's contribution may not be seen until well-after its publication, and it is encouraging that Self-Net has played a leading role in influencing several other works that were published later. Specifically, The paper 'Continual Learning with HyperNetworks' built upon many of the same ideas and proposed potential solutions to some of problems acknowledged in the original Self-Net paper [von Oswald et al. 2019a]. In the paper, Oswald Et al. trained a Hyper-Network [Ha et al. 2016] to map environment input data to weights

capable of solving the underlying problem, thereby providing a solution to the problem of task-inference mentioned in our original work. We applaud this contribution, and believe our investigative research into continual learning compression ratios may be of considerable assistance in determining optimal model capacity for specific task sequences.

CHAPTER 3

Deep Active Learning via Open-Set Recognition

3.1 Introduction

The goal of active learning is to infer the informativeness of unlabeled samples so as to minimize the number of requests to the oracle. Here, we formulate active learning as an open-set recognition problem. In this paradigm, only some of the inputs belong to known classes; the classifier must identify the rest as unknown. More specifically, we leverage variational neural networks (VNNs), which produce high-confidence (i.e., low-entropy) predictions only for inputs that closely resemble the training data. We use the inverse of this confidence measure to select the samples that the oracle should label. Intuitively, unlabeled samples that the VNN is uncertain about contain features that the network has not been exposed to; thus they are more informative for future training. We carried out an extensive evaluation of our novel, probabilistic formulation of active learning, achieving state-of-the-art results on MNIST, CIFAR-10, CIFAR-100, and Fashion-MNIST. Additionally, unlike current active learning methods, our algorithm can learn even in the presence of out-of-distribution outliers. As our experiments show, when the unlabeled pool consists of a mixture of samples from multiple datasets, our approach can automatically distinguish between samples from ***all*** seen vs. unseen datasets. Overall, our results show that high-quality uncertainty measures are key for pool-based active learning.

Supervised deep learning has achieved remarkable results across a variety of domains by leveraging large, labeled datasets [Lecun et al. 1998]. However, our ability to collect data

far outstrips our ability to label it, and this difference only continues to grow. This problem is especially stark in domains where acquiring the ground truth requires a highly trained specialist, e.g., medical imaging. Even in cases where labeled data is sufficient, there may be reasons to limit the amount of data used to train a model, e.g., time, financial constraints, or to minimize the model's carbon footprint.

Fortunately, the relationship between a model's performance and the amount of training data is not linear. There often exists a small subset of highly informative samples that can provide most of the information needed to learn to solve a task. In this case, we can achieve nearly the same performance by labeling (and training on) only those informative samples, rather than the entire dataset. The challenge, of course, is that the true usefulness of a sample can only be established a posteriori, after we have used it to train our model.

The growing field of active learning (AL) is concerned with automatically predicting which samples from an unlabeled dataset are most worth labeling [Sinha et al. 2019]. In the standard AL framework, a selector identifies an initial set of promising samples; these are then labeled by an oracle (e.g., a human expert) and used to train a task network [Gal et al. 2017]. The selector then progressively requests labels for additional batches of samples, up to either a percentage threshold (e.g., 40% of the total data) or until a performance target is met. In short, an active learning system seeks to construct the smallest possible training set which will produce the highest possible performance on the underlying task/s.

In this paper, we formulated active learning as an open-set recognition (OSR) problem, a generalization of the standard classification paradigm. In OSR, only some of the test

inputs are from the trained-upon distribution; the classifier must label the remaining inputs as out-of-distribution (OOD), meaning that they do not match the types of inputs it was trained on. For example, if a network was trained on digit recognition, e.g., using MNIST, then Images of animals or vehicles, such as those of CIFAR-10, would be OOD. Here, we view the labeled pool as the training distribution. The unlabeled samples which are similar to the labeled pool are deemed as in-distribution, while the unlabeled samples that are very different from the labeled pool are marked as OOD. Our hypothesis is that the samples most worth labeling are those that are most different from the currently labeled pool (i.e., those deemed OOD) because they contain features which the network has not yet been exposed to. Thus, training on these samples will allow the network to learn these features that are underrepresented in the existing training data. In short, our AL selection mechanism consists of picking unlabeled samples that are OOD relative to the labeled pool.

Figure 3.1 illustrates our proposed approach. In more detail, our classifier is a variational neural network (VNN) [Mundt et al. 2019b], which produces high-confidence (i.e., low-entropy) outputs only for inputs that are highly similar to the training set. VNNs are explicitly trained to maximize the entropy of their outputs for inputs that differ from the training set; thus, entropy-based confidence measures are more reliable for VNNs than for regular neural networks. Specifically, we use the inverse of this entropy-based confidence measure to select which unlabeled samples to query next. In other words, our selector requests labels for the samples that the classifier is least confident about because this implies that the existing training set does not contain items with similar features to them. As we

detail in section 4, our OSR-based approach achieved state-of-the-art results in a number of datasets and AL variations, far surpassing existing methods.

3.2 Prior Work

3.2.1 Pool-Based Active Learning

It has been shown that training samples do not contain equal amounts of useful information [Settles 2010]. Thus, the goal of sampling-based active learning is to learn an acquisition function that chooses the best data points for which a label should be requested from a large, unlabeled pool of data [Gal et al. 2017]. There have been numerous efforts to learn an optimal sampling strategy, and they can be broadly grouped into three major categories [Sinha et al. 2019].

3.2.1.1 Uncertainty-based Techniques

Uncertainty-based techniques aim to select samples from the unlabeled distribution about which the current classifier is highly uncertain. Different metrics have been proposed for quantifying how uncertain a model about a sample. Some methods such as Settles (2012) [Settles 2012], Settles and Craven (2008) [Settles and Craven 2008], and Luo et al. (2013) [Luo et al. 2013] used the entropy of the posterior probability of the model, whereas methods such as Joshi et al. (2009) and Roth and Small (2006) [Roth and Small 2006] use difference margin between the first and second predicted class to select the samples. Some approaches (Lewis and Catlett, 1994 [Lewis and Catlett 1994]; Lewis and Gale, 1994 [Lewis and Gale 1994]; Wang et al., 2016 [Wang et al. 2016]) directly use the probability outputs to select

the samples. Other methods map the network's outputs to a probability distribution to achieve better sample selection from the unlabeled pool. For example, Yoo and Kweon (2019) [Yoo and Kweon 2019] proposed a loss-learning module along with regular classifier which can predict the loss for given unlabeled pool image; Images with high prediction loss are selected to be labeled by the oracle. Similarly, Gal and Ghahramani (2016) [Gal and Ghahramani 2016] used a Monte Carlo dropout methods to obtain an uncertainty estimate for each sample.

3.2.1.2 Diversity and Hybrid-Based Methods

Representations-based models aim to maximize the diversity in training batches [Sener and Savarese 2017]. For example, Kirsch et al. (2019) [Kirsch et al. 2019] used a Bayesian formulation to determine sample diversity, while used gradient embeddings for assessing the similarity between samples. The approach in Shui et al. (2019) [Shui et al. 2020], on the other hand, formulate sample selection as distribution matching. Hybrid approaches attempt to combine quantifiable uncertainty and diversity in order to select training samples [Li and Guo 2013]. VAAL [Sinha et al. 2019] proposed an adversarial learning based method in which a discriminator is trained along with the task network to discriminate whether an example belongs to the labeled or unlabeled set. In Sener and Savarese (2017) [Sener and Savarese 2017], the authors considered active learning as a set-cover problem, one in which a task network is trained using a core-set loss, which is the difference between a task-network's classification error over the labeled set vs. the core-set. DBAL [Gal et al. 2017] approached the active learning problem using Bayesian convolutional neural networks, wherein confidence

is measured using variation ratios. In MC-Dropout [Gal and Ghahramani 2016], the authors proposed to model the uncertainty present in deep networks by interpreting dropout as a type of Bayesian inference in deep Gaussian processes.

3.2.2 Open-Set Recognition

Open-Set Recognition (OSR) refers to the ability of a system to distinguish between types of data it has already seen (the training distribution) from types to which it has not yet been exposed (out-of-distribution (OOD) data). Standard deep neural networks are not suitable for OSR because they often yield high confidence values for inputs which are significantly different from the training classes. This vulnerability has been exploited for adversarial attacks on deep networks, specifically to change a classifier's labels based on imperceptible changes to the input image (Goodfellow et al., 2014). VNNs, on the other hand, are explicitly trained to maximize the entropy of their outputs for samples that differ from those it was trained on, so they achieve OSR results.

More generally, as noted by Geng et al. (2020), existing OSR methods can be subdivided into discriminative-based and generative-based approaches. Discriminative methods modify traditional ML and deep neural networks to tackle the OSR problem. For example, Scheirer et al. (2012) used traditional SVMs with an additional open space risk term, while (Zhang and Patel, 2016) extended sparse classifiers to OSR by modeling the error distribution using Extreme Value Theory (EVT) (Vignotto and Engelke, 2018). Some other discriminative methods use nearest neighbors (Júnior et al., 2017), probability models (Jain et al., 2014; Scheirer et al., 2014; Scherreik and Rigling, 2016), or outlier detection (Bendale and Boult,

2015).

Generative methods primarily use generative adversarial networks (GANs) (Goodfellow, 2016) for OSR. For example, Neal et al. (2018) proposed G-OpenMax by adopting an encoder-decoder GAN architecture for generating samples which are highly similar to training samples yet do not belong to any of the training classes. Following a similar approach, Yang et al. (2019) investigated the open-set human activity recognition problem based on micro-Doppler signatures by using a GAN to generate samples which were highly similar to the target class and forming a negative set out of it. Not all generative approaches use GANs, though. For example, Geng and Chen (2018) proposed a collective, decision-based OSR model by slightly modifying the hierarchical Dirichlet process.

3.3 Methodology

As noted above, our active learning approach iteratively selects samples from an unlabeled pool based on the confidence level of its OSR classifier. Below, we first formalize the active learning paradigm we are tackling, then detail our proposed system. In particular, we provide an overview of VNNs and explain how we use their outputs to select new samples to label.

3.3.1 Formal problem definition

Formally, an active learning problem is denoted as $P = (C, D_{train}, D_{eval})$, where C indicates the number of classes, D_{train} is the training set, and D_{eval} is the evaluation set, s.t. $D_{train} \cap D_{eval} = \emptyset$.

Let $\mathcal{D}_{train} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ be a dataset consisting of N i.i.d. data points where only

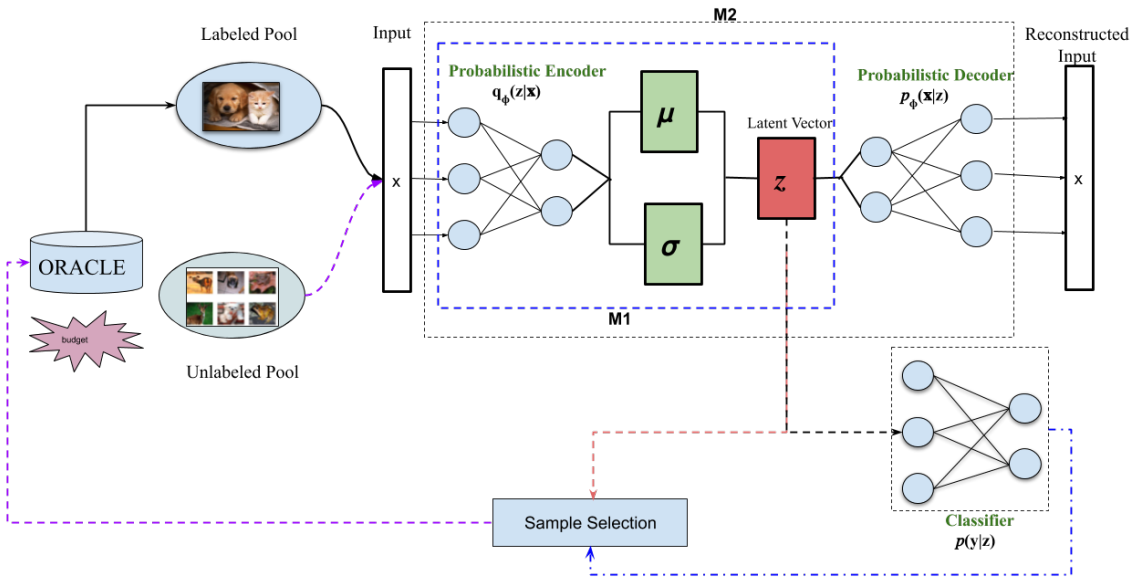


Figure 3.1 Framework overview: Our proposed active learning system uses open-set recognition to identify which samples from the unlabeled pool to label. Our classifier is a variational neural network (VNN) [Mundt et al. 2019b], which simultaneously reconstructs an input using a probabilistic autoencoder (AE) and classifies it by feeding the AE's latent vector z to a linear classifier. We use the VNN's loss function to determine which samples to select from the unlabeled pool (Sample Selection). As in [Mundt et al. 2019b], we tested two VNN variants: M1 is trained using only the loss on the latent vector $q_\Phi(z|x)$ and the classifier $p(y|z)$, while M2 also includes the loss on the reconstructed input $p_\Phi(x|z)$. Figure based on similar diagrams in [Mundt et al. 2019a] and [Sinha et al. 2019].

m of them are labeled ($m \ll N$). Each sample $\boldsymbol{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector, and $y_i \in \{1, 2, \ldots, C\}$ represents the target label. At the start, $\mathcal{D}_{train}$ is partitioned into two disjoint subsets: a labeled set $\mathcal{L}$ which consists of the m labeled data points, and an unlabeled set $\mathcal{U}$ which consists of the remaining $N - m$ data points with unknown target labels. We will update both $\mathcal{L}$ and $\mathcal{U}$ after each iteration of our algorithm. We denote the state of a subset at a given timestep as $\mathcal{L}^t$ and $\mathcal{U}^t$, respectively, for $t \in \{0, 1, \ldots\}$.

In active learning, we first train a classifier f, with parameters θ, on $\mathcal{L}^0$. Afterwards we select b data points from $\mathcal{U}^0$ using our OSR criterion (see Sec. 3.3.2). These b data points

are then sent to the oracle for annotation. The annotated samples are removed from the unlabeled pool and added to the labeled pool, along with their newly acquired target labels. The updated labeled and unlabeled data pools become $\mathcal{L}^1$, of size $m+b$, and $\mathcal{U}^1$, respectively. Thus, the labeled pool grows in size as training progresses. We continue this process until the size of the labeled pool reaches a predefined limit (40% of D_{train} in our experiments).

Importantly, unlike other formulations of AL, we allow for the unlabeled pool $\mathcal{U}$ to contain training data from *multiple datasets*. As we show in our experiments, our OSR-based AL method can automatically ignore samples that do not belong to the target classes.

Algorithm 2 Active Learning

Input: Unlabeled pool $\mathcal{U}^0$, labeled pool $\mathcal{L}^0$ for $t \in \{0, 1, \ldots\}$ where size of $\mathcal{L}^0 = m_0$.
Require: Active Learning Model, Optimizer, Sampling Strategy
Require: initialize b (budget), θ (Model parameters), Epochs
repeat
  Train Active Learning Model on Labeled Pool ($\mathcal{L}^t$) using selected optimizer.
  Give trained model f_θ on Labeled Pool ($\mathcal{L}^t$), Sampling Strategy (3.3.3 or 3.3.4) selects the uncertain data points according to budget size b.
  Send the selected data points to Oracle for annotation.
  Add the annotated data points to the Labeled Pool ($\mathcal{L}^t$)
until *stopping criterion (size of Labeled Pool* ($\mathcal{L}^t$) *equals 40% of* D_{train}*)*;

3.3.2 Active learning system

Algorithm 2 summarizes our AL approach, which has two main components: a variational neural network (VNN) [Mundt et al. 2019b] that serves as our classifier and an OSR selection mechanism based on the loss function of the VNN. We discuss each component below.

3.3.2.1 Variational Neural Networks (VNNs)

Variational neural networks (VNNs) [Mundt et al. 2019b] are a supervised variant of β-variational autoencoders (β-VAE) [Higgins et al. 2017]. The latter is itself a variant of VAEs [Doersch 2016] but with a regularized cost function. That is, the cost function for a β-VAE consists of two terms: the reconstruction error, as with a regular VAE, and an *entanglement* penalty on the latent vector. This penalty forces the dimensions of the latent space to be as uncorrelated as possible, making them easier to interpret.

A VNN combines the encoder-decoder architecture of a β-VAE with a probabilistic linear classifier (see Fig. 2.1 for a visual representation). As such, its loss function includes a classification error, i.e., a supervised signal, in addition to the reconstruction and entanglement terms:

$$\begin{aligned} L(\theta, \phi, \xi) = \mathbb{E}_{q_\theta(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_\phi(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\xi}}(\boldsymbol{y}|\boldsymbol{z})\right] \\ - \beta \, \mathrm{KL} \left(q_\theta(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right) \end{aligned} \tag{3.1}$$

As detailed in [Mundt et al. 2019b], θ, ϕ, and ξ are the parameters of the encoder, decoder, and classifier, resp., while $p_\phi(\boldsymbol{x}|\boldsymbol{z})$ and $p_{\boldsymbol{\xi}}(\boldsymbol{y}|\boldsymbol{z})$ are the reconstruction and classification terms. The last term is the entanglement penalty, which is given by the Kullback-Leibler divergence between the latent vector distribution and an isotropic Gaussian distribution.

As in [Mundt et al. 2019b], we evaluated both the full framework discussed above (dubbed M_2 in our experiments), which uses the loss function in Eq. 3.1, and a simplified version (M_1)

without the reconstruction error:

$$L(\theta, \xi) = \mathbb{E}_{q_\theta(\boldsymbol{z}|\boldsymbol{x})} \left[\log p_{\boldsymbol{\xi}}(\boldsymbol{y}|\boldsymbol{z})\right] - \beta \, \mathrm{KL} \left(q_\theta(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\right) \tag{3.2}$$

As our experiments show, both versions outperform the state of the art, but M_2 achieves better results overall.

3.3.2.2 Sample Selection

We wish to leverage the class disentanglement penalty defined in Eq. 3.1. Specifically, our aim is to select b data points from the unlabeled pool $\mathcal{U}$ that the VNN is highly uncertain about. Following [Mundt et al. 2019a], in our experiments we investigated two sampling algorithms for OSR: *uncertainty sampling* and *Weibull distribution sampling*. The former is simpler, but the latter allows one to better reject outliers. We briefly describe each sampling strategy below.

3.3.3 Uncertainty sampling

Here, we select a data point $\boldsymbol{x}_i$ based directly on how uncertain the VNN is about it. Specifically, we rank all unlabeled samples by the value of the most likely class label and select the b samples with the lowest maximum values. Since the sum of class likelihoods is normalized, the value of the maximum class probability will approach one for highly certain samples and approach $\frac{1}{|C|}$, where $|C|$ is the number of classes, for highly uncertain samples. In other words, the class likelihoods of uncertain samples have higher entropy than those for which the VNN is certain about.

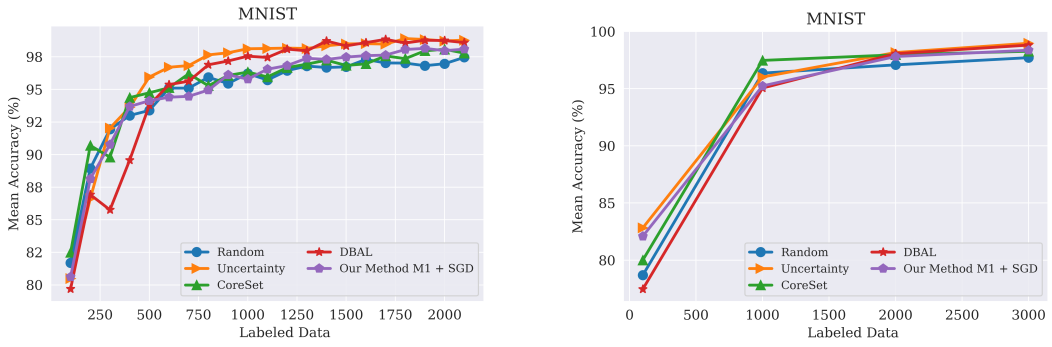
3.3.4 Wiebull distribution sampling

As our experiments show, uncertainty sampling is suitable for active learning problems in which all unlabeled samples belong to known classes. However, for the case where the unlabeled pool also contains samples from unknown classes, we need a more robust way to exclude outliers. For this latter case, we employed the sampling procedure defined in [Mundt et al. 2019a], which leverages a Wiebull distribution to estimate the model's uncertainty w.r.t a specific sample.

For completeness, here we will briefly outline the methodology proposed in [Mundt et al. 2019a]. Intuitively, it can be shown that it is useful to quantify the probability that a given data sample is an outlier, herein defined as a sample which is not sufficiently similar to those which have already been correctly classified. [Mundt et al. 2019a] show that this can be accomplished as follows. First, for each class, we compute the mean of the latent vectors of all samples that have been correctly predicted by the model. Second, we compute the distances from each class mean for all latent vectors, which [Mundt et al. 2019a] showed can be modeled with a Wiebull distribution. As such, a sample's likelihood under this distribution constitutes the minimum probability that the sample does ***not*** belong to any previously known class. In other words, the lower this value, the more likely that the sample is an outlier.

3.4 Experimental Results

We performed experiments on three image classification datasets—MNIST, CIFAR-10, and CIFAR-100—following the methodology defined in Section 3.3. Below, we first present our implementation details, then discuss our results.



(a) (b)

Figure 3.2 Performance on MNIST classification tasks using different query sizes for model M_1. (a) Query batch size of 100; (b) Query batch size of 1000 compared to Core-set [Sener and Savarese 2017], DBAL [Gal et al. 2017], Random Sampling and Uncertainty Sampling. M1 indicates our model with Encoder and Classifier. Best visible in color. Prior results adapted from [Sinha et al. 2019].

3.4.1 Implementation Details

Budget: For CIFAR-10 and CIFAR-100, we used a max budget of 40%, and stage budgets b of 10%, 15%, 20%, 25%, 30%, 35%, and 40%. For MNIST, we used stage budgets of 100 and 1000 Images.

Runs: For all three datasets, we measured performance by computing the average accuracy across 5 independent runs.

State of the art comparison: We compared our method against several recent AL

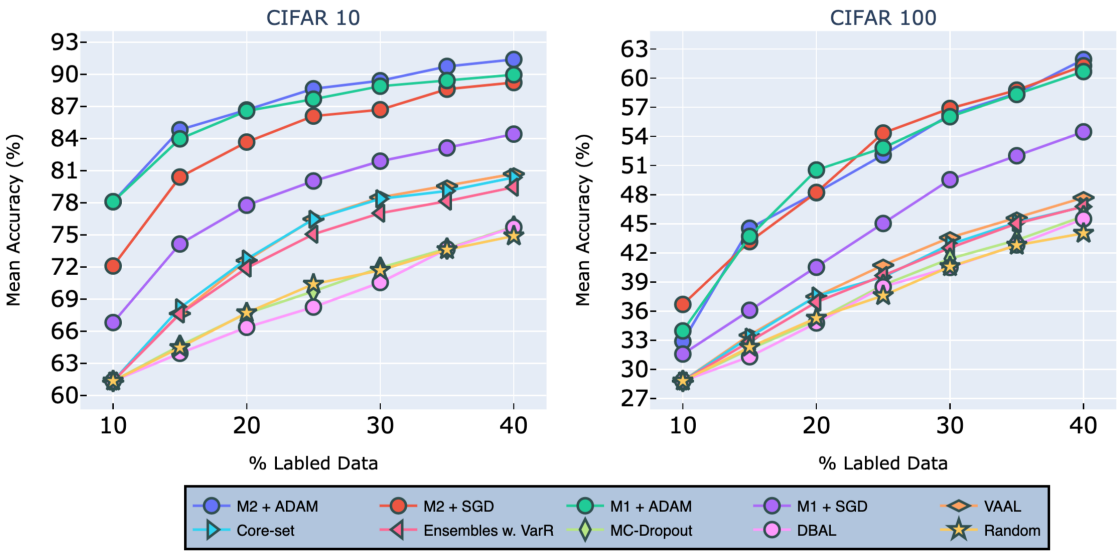


Figure 3.3 Performance on classification tasks for CIFAR-10 (left) and CIFAR-100 (right) compared to VAAL [Sinha et al. 2019], Core-set [Sener and Savarese 2017], Ensembles w. VarR [Beluch et al. 2018], MC-Dropout [Gal and Ghahramani 2016], DBAL [Gal et al. 2017], and Random Sampling. M1 indicates our model (3.2) and M2 indicates our model (3.1). All the legend names are in descending order of final accuracies. Best visible in color. Prior results adapted from [Sinha et al. 2019].

approaches including Variational Adversarial Active Learning (VAAL) [Sinha et al. 2019], Core-Set [Sener and Savarese 2017], Monte-Carlo Dropout [Gal and Ghahramani 2016], Ensembles using Variation Ratios (Ensembles w. VarR) [Beluch et al. 2018; Freeman 1965], and Deep Bayesian AL (DBAL) [Gal et al. 2017]. As a baseline, we also included uniform random sampling (Random) since it remains a competitive strategy in the field of active learning.

Architectures: For experiments on CIFAR-10 and CIFAR-100 we used a VGG16 network [Simonyan and Zisserman 2014] as the encoder for both models, M_1 and M_2, and a decoder based on 14-layer residual networks [Higgins et al. 2017; Zagoruyko and Komodakis 2016]. We used latent vectors of size 60. As noted in Sec. 3.3, the classifier consists of a

single linear layer. For MNIST, we used a LeNET network [Lecun et al. 1998] as our encoder and a latent vector of size 60.

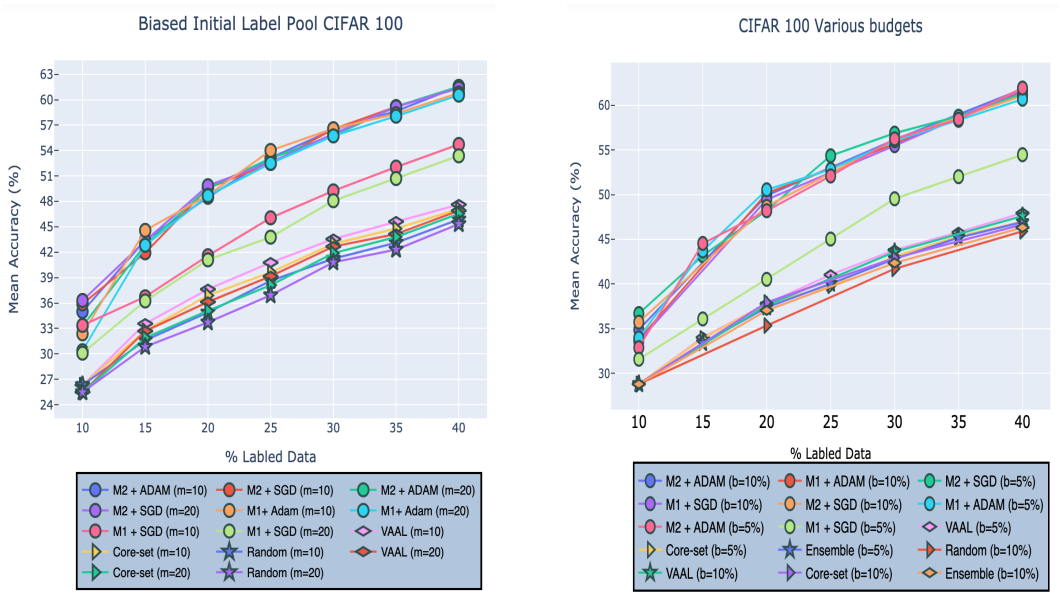
Optimization: We optimized all models using a mini-batch size of 128, a learning rate of 0.001, and a weight decay of 10^{-5}. We tested two different optimizer, SGD and ADAM [Kingma and Ba 2014], for both M_1 and M_2, for a total of four combinations:

- M_1^{sgd} - Model M_1 as shown in Eq. 3.2 with SGD optimizer.
- M_1^{adam} - Model M_1 as shown in Eq. 3.2 with Adam optimizer.
- M_2^{sgd} - Model M_2 as shown in Eq.3.1, with SGD optimizer.
- M_2^{adam} - Model M_2 as shown in Eq.3.1 with Adam optimizer.

Oracle queries: We defined a learning stage (i.e., a period of training between queries to the oracle) as lasting 150 epochs on CIFAR-10 and CIFAR-100 and 10 epochs on MNIST. At the completion of a stage, we requested labels for b Images from the unlabeled pool. These were added to the labeled pool and used in the subsequent learning stages.

3.4.2 Image classification results

MNIST: Our results were comparable with the state of the art on MNIST. However, as Figs. 3.2(a) and Fig. 3.2(b) show, random sampling is already a highly successful strategy on MNIST, leaving little room for improvement on this dataset. In particular, as illustrated in Fig. 3.2(b), all methods obtained statistically similar results as the batch size increased. However, as shown in Fig. 3.2(a) methods such as DBAL or Coreset have lower accuracies at the initial stages when using smaller batch sizes.



(a) (b)

Figure 3.4 Robustness of our approach on CIFAR-100 given (a) biased initial labeled pool or (b) different budget sizes compared to VAAL [Sinha et al. 2019], Core-set [Sener and Savarese 2017] , Ensembles w. VarR [Beluch et al. 2018], MC-Dropout [Gal and Ghahramani 2016], DBAL [Gal et al. 2017], and Random Sampling. M1 indicates our model (3.2) and M2 indicates our model (3.1). Best visible in color. Prior results adapted from [Sinha et al. 2019].

CIFAR-10 & CIFAR-100: As Fig. 3.3 clearly shows, we achieved state-of-the-art performance by a considerable margin on both CIFAR-10 (left) and CIFAR-100 (right).

On CIFAR-10, models $[M_1^{sgd}, M_1^{adam}, M_2^{sgd}, M_2^{adam}]$ achieved mean accuracies of [84.4%, 89.24%, 89.97%, 91.4%], respectively. To put this in perspective, the original accuracy for this VNN using the entire CIFAR-10 dataset was 92.63%. VAAL came in second, with an accuracy of only 80.71% , followed by Core-Set with an accuracy of 80.37%, and then Ensemble w VarR at 79.465%. Random sampling, DBAL and MC-Dropout all trailed significantly behind other methods. Finally, we found that our models trained with ADAM, on average, outperform those trained with SGD.

On CIFAR-100, models $[M_1^{sgd}, M_1^{adam}, M_2^{sgd}, M_2^{adam}]$ achieved mean accuracies of [54.47%, 60.68%, 61.25%, 61.93%], resp. The original accuracy with the entire CIFAR-100 dataset was 63.14%. VAAL once again came in second, with an accuracy of 54.47 %, followed by Core-Set, and Ensemble w VarR.

3.4.3 Additional experiments

In addition to our classification experiments, we replicated and extended the experiments of the same name put forth in [Sinha et al. 2019] in order to investigate the robustness of our approach. Unless otherwise stated, we used CIFAR-100 for these experiments. Finally, we also tested our methods' ability to learn when the unlabeled pool contained out-of-distribution samples, a case which, to the best of our knowledge, cannot be handled by any existing methods.

Effect of Biased Initial Pool: We first investigated the effect of bias that may be present in the initial labeled pool, $\mathcal{L}_0$. As stated in [Sinha et al. 2019], bias can negatively impact the training of an active learner because it means that the initial labeled pool may not be representative of the true underlying data distribution. Unless explicitly accounted for, this will cause a system to learn an incomplete, or biased, model of the latent space. Following the protocol defined in [Sinha et al. 2019], we removed all data points for c classes from $\mathcal{L}_0$, thereby unbalancing the dataset and thus introducing bias. As shown in Fig. 3.4(a), our method outperformed VAAL, Core-set, and random sampling w.r.t selecting useful data points from classes that were underrepresented in the initial labeled pool. Models $[M_1^{sgd}, M_1^{adam}, M_2^{sgd}, M_2^{adam}]$ achieved accuracies of [53.35%, 60.54%, 61.36%, 61.55%], re-

spectively, when $c = 20$ and [54.72%, 60.79%, 61.53%, 61.57] when $c = 10$ (as noted above, c is the number of classes from which to exclude data). VAAL, by comparison, came in second, followed by Core-set, exhibiting accuracies [46.91%, 46.55%] for c=20 and [47.10%, 47.63%] for c=20, respectively. Random sampling achieved an accuracy of 45.33% for $c = 10$ and 45.87% for $c = 20$.

Effect of Budget Size on Performance: In this section, we tested the effect of different budget sizes b on performance. Specifically, we investigated the effect of budgets of size $b = 5\%$ and $b = 10\%$, referring to percentage of samples taken from $\mathcal{D}_{train}$ at each stage of learning. As shown in Fig. 3.4(b), our model outperformed VAAL, Core-Set, Ensemble, and random sampling over both the budget sizes. VAAL comes in second followed by Core-set and Ensemble. Models $[M_1^{sgd}, M_1^{adam}, M_2^{sgd}, M_2^{adam}]$ achieve accuracies of [61.52%, 61.57%, 61.07%, 61.82%] for $b = 10$ and [54.32%, 60.68%, 61.29%, 61.9%] for $b = 20$.

Noisy Oracle: Next, we investigated the performance of our approach in the presence of noisy data caused by an inaccurate, or noisy oracle. As in [Sinha et al. 2019], we assumed that incorrect labels can be caused by the natural ambiguity which exists between examples drawn from 2 separate classes, rather than adversarial attacks. CIFAR-100 has both classes and super-classes, so, following [Sinha et al. 2019], we randomly modified the labels of either 10%, 20% or 30% of the samples by replacing them with a label from another class within the same super-class. As shown in Fig. 3.5, our models consistently outperformed existing approaches *across all noise levels*. In other words, our M_1 model with 30% noise was *more accurate* than VAAL, etc. with 10% noise.

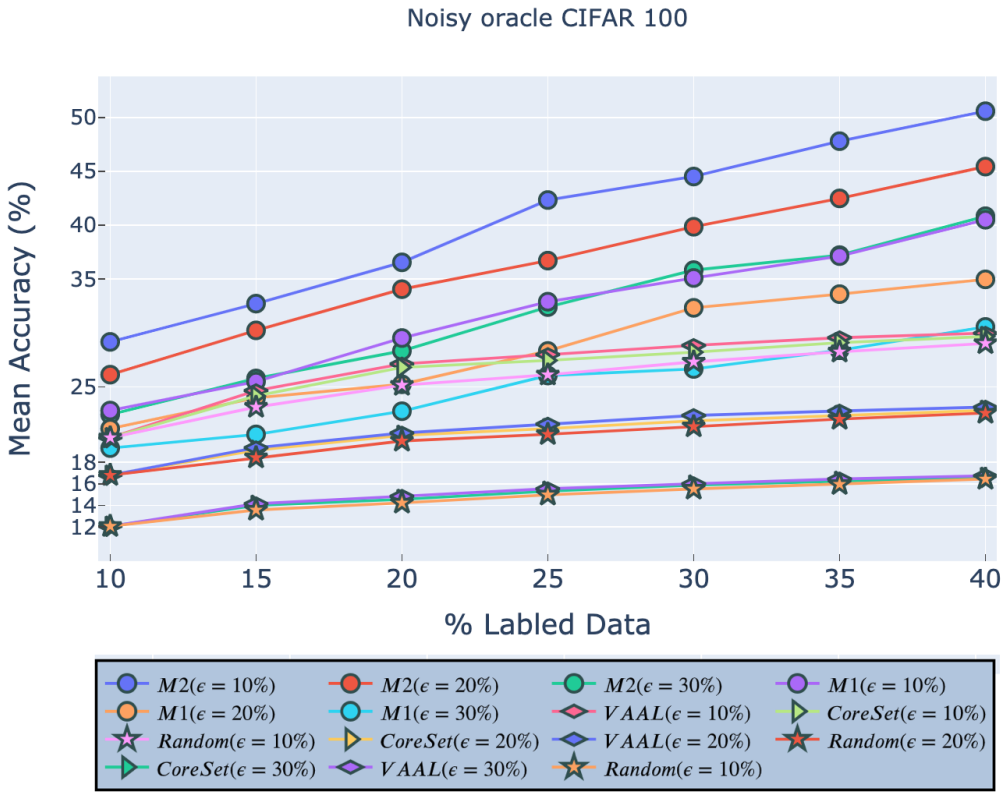


Figure 3.5 Robustness of our approach on CIFAR-100 given a noisy oracle. M_1 indicates our model (3.2) and M_2 indicates our model (3.1). All legend names are in descending order of final accuracies.

Sampling Time Analysis We also replicated the sampling time analysis put forth in [Sinha et al. 2019]. Table 3.1 shows that our method is competitive with other state-of-the-art techniques w.r.t. execution time, thereby offering strong empirical evidence that our method offers large performance advantages with minimal additional computation.

Out-of-distribution samples in unlabeled pool: Finally, we also tested an extreme case of active learning in which data samples from other datasets are mixed into the current unlabeled pool. We used CIFAR-10 for these experiments. Here, we intentionally added 20% data (10,000 Images) from other datasets to the unlabeled pool; thus, the network must

Table 3.1 Sampling Time Analysis: Mean time to select a sample from the unlabeled pool of CIFAR-100.

Method	Time (Seconds)
VAAL	10.69
Uncertainty sampling	**10.89**
DBAL	11.05
Weibull sampling	**20.41**
Ensembles w. VarR	20.48
Core-set	75.33
MC-Dropout	83.65

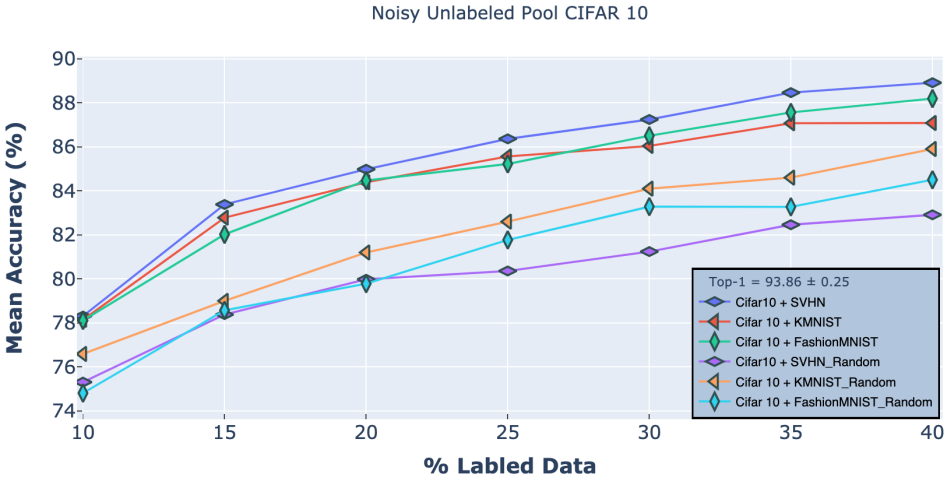


Figure 3.6 Robustness of our approach on CIFAR10 classification tasks when the unlabeled pool includes samples from either the SVHN, KMNIST, or FashionMNIST datasets. The first three curves used the M_2 classifier, while the ones with the 'Random' subscript used random sampling. Our results confirm that our approach significantly outperforms this baseline.

distinguish not only between informative and non-informative samples but also distinguish *in-distribution* data samples from *out-of-distribution* samples. Whenever our model selected an OOD sample, the oracle discarded the sample, thus reducing the overall budget size. The discarded samples were placed back in the unlabeled pool (so the total number of OOD samples remained at 10,000).

Figure 3.6 shows our M_2 method's performance on CIFAR-10 when the unlabeled pool

contained Images from either SVHN, KMNIST, or FashionMNIST. Here, we used Weibull sampling (Sec. 3.3.4) due to its better outlier rejection compared to uncertainty sampling. For comparison, we also tested random sampling as a baseline. Impressively, despite the presence of 20% OOD samples, our method significantly outperformed existing state-of-the-art methods trained on the regular unlabeled pool (Fig. 3.3). And its performance, regardless of the second dataset, was only slightly below the standard M_2 method.

3.5 Conclusions and Future work

We have presented a novel approach for deep active learning using open-set recognition. To the best of our knowledge, we are the first to merge AL with OSR. Extensive experiments conducted over several image classification datasets have verified the effectiveness of our approach and established new state-of-the-art benchmarks. Specifically, we empirically demonstrated that the samples most worth labeling are those which are most different from the current labeled pool. Training on such samples allows the model to learn features underrepresented in the existing training data. We extensively tested the robustness of our approach using different budget sizes, a noisy oracle, and an unlabeled pool comprised of multiple datasets. In future work, we plan to test our approach on continual learning problems, in which the system must learn to solve different problems over time. We also plan to test our method on other problems, including image segmentation and document classification.

CHAPTER 4

Deep Active Learning with Barlow Twins

4.1 Introduction

The generalization performance of a Deep Neural Network is heavily influenced by the quantity, quality, and diversity of the data upon which it is trained. The acquisition of this data is often straightforward, but labeling can be expensive and time-consuming. As such, the goal of Active Learning is to select the most informative training samples from an unlabeled dataset for subsequent annotation. Concurrently, Self-Supervised Learning (SSL) has gained significant popularity by narrowing the performance gap with supervised methods on large-scale computer vision benchmarks. SSL has demonstrated the ability to generate low-level representations that remain invariant to various input distortions, such as rotation, solarization, cropping, and more. Interestingly, SSL approaches often rely on simpler and more scalable learning frameworks. In this paper, we propose Deep Active Learning with Barlow Twins (DALBT), a novel Active Learning method that leverages the power of SSL. We achieve this by training a simple classifier alongside a Self-Supervised learner, called Barlow Twins, to create an encoder that is invariant to artificially created distortions. DALBT stands out for its simplicity, ease of implementation, and broad applicability. Additionally, building off our prior work, we employ Weibull sampling to select samples for labeling. We conduct extensive evaluations of our novel approach, achieving state-of-the-art results on MNIST, Fashion-MNIST, and CIFAR-10 datasets. Furthermore, we demonstrate the model's robustness by evaluating it on a mixed pool of samples from *multiple datasets*,

successfully detecting and quantifying the novelty of *entire* datasets.

While Deep Neural Networks (DNNs) have demonstrated state-of-the-art accuracy in supervised learning tasks like classification, object detection, and semantic segmentation, they typically require large labeled datasets. The challenges of labeling massive datasets in real-world scenarios are numerous and include high costs, limited time availability from domain experts, lengthy labeling requirements for large-scale data such as videos and time-series data, financial constraints, and the desire to minimize carbon footprints. These limitations restrict the application of DNNs to various research areas and organizations. To address these challenges, Active Learning (AL) systems aim to maximize model performance under a fixed training budget. It therefore highlights a nonlinear relationship between a model's performance and the amount of training data it needs, thus emphasizing the importance of selecting a *good* subset of unlabeled data for future annotation.

In contrast, Self-Supervised Learning (SSL) learns useful information from datasets without relying on human annotations. SSL focuses on obtaining high-quality low-level representations of input data without access to target labels. In fact, recent advancements in SSL have brought it closer to supervised learning methods in terms of performance on large datasets and computer vision tasks. At a high level, both Active Learning and Self-Supervised Learning share the goal of reducing the labeling requirements. To merge these approaches, our method incorporates both Self-Supervised Learning and Supervised Learning at each step of the Active Learning process. This differs from previous approaches that first pre-train on the entire unlabeled dataset using Self-Supervised Learning and then proceed with Ac-

tive Learning using Supervised Learning, a process that can be time-consuming for large unlabeled datasets.

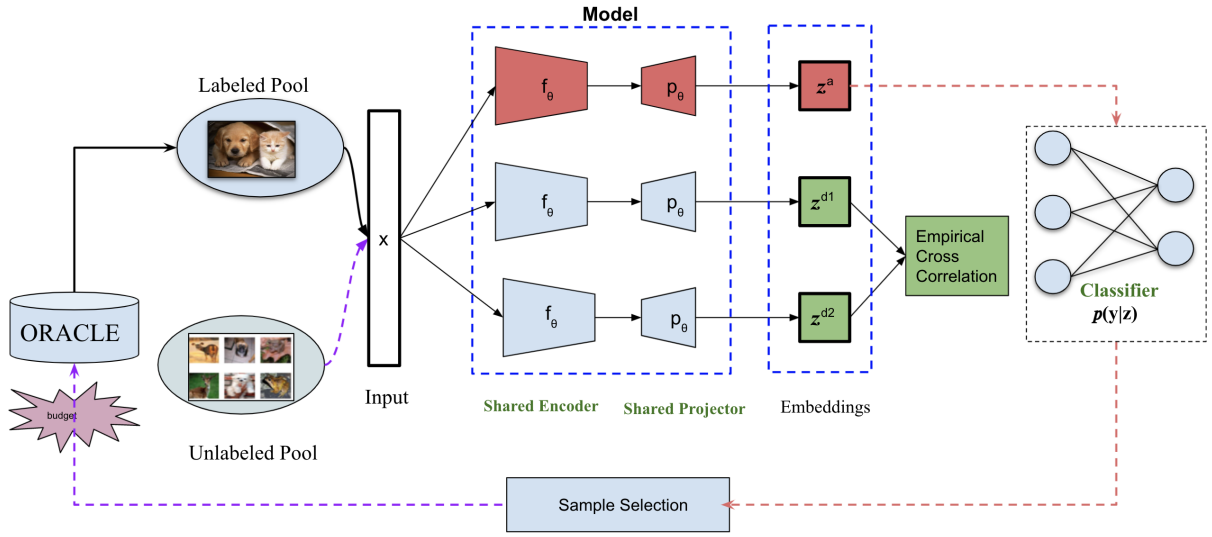


Figure 4.1 **DALBT Framework overview:** Our proposed active learning system. It consists of an encoeder(E), a Projector(P), and a Classifer(C). The Encoder consumes two distorted versions of same input sample. The output of the Encoder is then fed into the Projector network, which projects both distorted versions into lower dimensional representations. The Classifier then takes the latent vector(z), produced by passing the raw input (non-distored) image through the Encoder(E). The system is trained end-to-end using a joint loss, simple the cross-corelation loss and the classification loss. Lastly, Wiebull Sampling is employed to identify which samples to label.

4.2 Related Work

Here we detail previous work done in each of these directions.

4.2.1 Active Learning

Active learning methodologies have recently been comprehensively reviewed by Dasgupta [2011]; Hanneke et al. [2014]; Settles [2010]. As these works explain, sufficiently informative

data-points contribute more to model performance than other less-informative training samples. Therefore, the primary objective of active learning is to learn an acquisition function to select the best unlabeled data points for which labels should be requested.

Existing active learning approaches can be divided into two categories: Pool-based methods and Query Synthesizing methods. Pool-based methods aim to identify the most informative samples from the unlabeled data using various sampling strategies, which are discussed in more detail in this section. In contrast, Query Synthesizing methods [Mahapatra et al. [2021]; McCallum and Nigam [1998]; Zhu and Bento [2017]] employ generative models to generate informative samples.

Active learning sampling strategies can be further subdivided into the following categories: (a) Uncertainty Sampling: This is one of the popular sampling methodologies whereby the model queries data points about which it is most uncertain. Recent research has shown that uncertainty sampling approaches have proven effective for deep learning models such as CNNs [Beluch et al. [2018]; Gorriz et al. [2017]; Scheffer et al. [2001]; Wang et al. [2017]]; (b) Diversity Sampling: This sampling method aims to select samples that are sufficiently diverse when compered to existing labeled samples. (c) Representative Sampling: This sampling method aims to choose samples from the unlabeled pool that are most representative of the entire dataset. Some research combines features from these three distinct groups to enhance the performance of active learning systems.

Schohn and Cohn [2000] presented an active learning algorithm using support vector machines. The key idea is to select informative unlabeled examples to label that are close to

the decision boundary. This is done by using the version space, which represents the set of possible hypotheses consistent with the labeled data. The algorithm selects unlabeled examples that minimize the version space, meaning they rule out the most possible hypotheses. Experiments on real-world datasets show SVMActive needs significantly less labeled data than uncertainty sampling to achieve the same accuracy.

Tong and Koller [2001] applied active learning to text classification. They used SVMs to implicitly project the training data into a different (often higher dimensional) feature space that is linearly separable. They then formulated the query selection problem as a version space optimization problem, optimizing the version space as quickly as possible while still adhering to SVM constraints, thereby finding informative samples quickly.

Tur et al. [2005] combines active and semi-supervised learning methods in the domain of spoken language understanding.

Wang et al. [2017] combines an uncertainty-based active learning algorithm with a diversity constraint using sparse selection, whereby sample selection is represented as a sparse modeling problem.

Sener and Savarese [2017] reformulated the problem of active learning as a core-set selection problem, whereby a set of points is chosen so that the selected points are *dissimilar* from each other and the labeled set.

Zhu et al. [2009] combined uncertainty and density (SUD) along with density-based reranking to address the problem of outlier selection. This approach selects samples that are not only the most informative (with respect to uncertainty) but also the most representative

with respect to density.

Similar to the core-set approach of selecting batches of images in a pool-based setting, Geifman and El-Yaniv [2017] selects points for each class using the farthest-first (FF) traversal principle, also known as the Gonzalez algorithm [[Gonzalez 1985]]. The FF principle involves selecting the first point 'x' randomly and then selecting the next point, which is farthest from the previously chosen point 'x,' by greedily choosing the point farthest from any of the points already selected. A set of points is obtained using neural activation over a representation layer by forward-passing all the unlabeled data. The farthest-first (FF) traversal principle is similar to constructing a long-tail Weibull distribution.

Gissin and Shalev-Shwartz [2019] is motivated by selecting samples for which the probability of distinguishing between the unlabeled pool and the labeled pool is the highest. Selecting such samples for labeling should be informative and improve the model's performance.

Beluch et al. [2018] demonstrated that ensembles perform better and lead to more calibrated predictive uncertainties, which can be used for Active Learning Uncertainty strategies. The authors also showed that this method outperforms Monte-Carlo Dropout and geometric approaches.

4.2.2 Self-Supervised Learning

In recent years, self-supervised learning has attained performance levels comparable to supervised learning. Most self-supervised learning methods aim to learn representations that remain invariant despite distortions present in the input data. These distorted inputs are

generated by applying various data augmentations randomly to the input. Different research methodologies strive to achieve this objective through a variety of approaches. For instance, in SIMCLR [Chen et al. 2020], this is accomplished by forming 'positive' and 'negative' sample pairs from the input data and treating each pair differently within the loss function. BarlowTwins [Zbontar et al. 2021] achieves this by utilizing variance and invariance terms, where two distorted versions of a single sample should produce a low-level representation. This is achieved using a custom loss function comprising variance and redundancy reduction terms.

Previous studies of the intersection between active learning and self-supervised learning have been explored in various domains. In the graphical domain, researchers applied self-supervised learning alongside active learning to Graph Neural Networks [Zhu et al. 2020]. They considered the information propagation scheme of a Graph Neural Network (GNN) and selected central nodes from homophilous ego networks . In another study, autoencoder architectures and the SSL technique SIMCLR were used to create positive and negative pairs [Bengar et al. 2021]. In NLP, for text classification tasks, self-supervised learning served as a pre-training step for training language models [Yuan et al. 2020]. In that work, samples for which the language model exhibited uncertainty were sent for labeling, enabling efficient fine-tuning. This work was followed by Wang et al. [2021] which employed *disfluency* detection, relying heavily on human-annotated data for sentence classification. In Bengar et al. [2021], the model was trained on the entire dataset to obtain a frozen backbone. Subsequently, a linear classifier or an SVM decoder was fine-tuned on top of the features

in a supervised manner. Inference was then executed on the entire unlabeled data, and the top-k samples were selected via an acquisition function. In the medical domain, researchers collected saliency maps of medical images and framed it as a self-supervised learning problem [Mahapatra et al. 2021]. An autoencoder was used to reconstruct the saliency maps, followed by clustering the latent space to collect the top-k samples and query labels of the most representative sample per cluster.

Our work stands out in the field due to its distinctive approach. Unlike previous research that heavily focuses on extensive pre-training on all the data, our method is pragmatic, especially given the substantial size of the unlabeled pool, which would normally introduce a significant training overhead. Additionally, creating "positive" and "negative" pairs for training becomes impractical when dealing with large overall dataset sizes.

4.3 Methodology

In this section, we will provide a concise overview of the setup for pool-based active learning in computer vision classification tasks. Additionally, we will delve into the self-supervised learning approach, Barlow Twins, and the underlying rationale for its utilization. Subsequently, we will introduce our novel approach, Deep Active Learning using Barlow Twins (DALBT). Throughout the following sections, we will consistently refer to the model under training as f, with its associated weights and parameters denoted as θ. Within each sampling iteration, our sampling method will carefully choose a diverse selection of examples from an unlabeled pool ($\mathcal{U}$) consisting of unlabeled examples denoted as $X_{\mathcal{U}}$. These selected

examples are chosen based on the model's uncertainty, ensuring their utility for training in the subsequent active learning iteration.

4.3.1 Problem Definition

Formally, the pool-based active learning problem can be defined in the following way. Let $P = (\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$, where $\mathcal{D}_{\text{train}}$ represents the training set from which the initial pool of samples is drawn. $\mathcal{D}_{\text{train}}$ can be further divided into two subsets, namely $(\mathcal{D}_{\text{L}}, \mathcal{D}_{\text{U}})$. Here, $\mathcal{D}_{\text{L}}$ denotes the labeled pool, where each sample is represented as a pair of input and label, denoted as $(x_{\text{L}}, y_{\text{L}})$. On the other hand, $\mathcal{D}_{\text{U}}$ represents a considerably larger pool of samples which are yet to be labeled.

The primary objective of the active learning model is to train on the labeled pool $\mathcal{D}_{\text{L}}$ and, in conjunction with a sampling method, iteratively select the most useful samples from the unlabeled pool $\mathcal{D}_{\text{U}}$ for annotation by an oracle. This process aims to minimize the expected loss within a fixed sampling budget b, where b represents the total number of informative samples that can be chosen from the unlabeled pool at each stage of the active learning setup. These selected b samples are subsequently forwarded to the oracle for annotation. We denote the state of a subset at a given timestep as $\mathcal{L}^t$ and $\mathcal{U}^t$, respectively, for $t \in 0, 1, \ldots$, where t signifies the current stage within the active learning process.

In the standard pool-based active learning setup, we initially train our active learning model (f) with parameters θ using the labeled pool at stage $t = 0$, denoted as $\mathcal{L}^0$. Following this initial stage $(t = 0)$, we select b data points from the unlabeled pool $(\mathcal{U}^0)$ using a predefined sampling method, such as the uncertainty measures or confidence estimates

describe above. These chosen b data points are then removed from the unlabeled pool ($\mathcal{U}^0$) and forwarded to the oracle ($\mathcal{O}$) for annotation. Subsequently, these annotated data points are added to the labeled pool ($\mathcal{L}^0$), which transforms into the labeled pool ($\mathcal{L}^1$), while the unlabeled pool ($\mathcal{U}^0$) transitions into ($\mathcal{U}^1$).

The model is then retrained on the new labeled pool ($\mathcal{L}^1$) in the subsequent stage, marked as $t = 1$. In the current experimental setup, we explore two scenarios: one where the unlabeled pool ($\mathcal{U}$) consists of samples from the same dataset, and another where it contains a mixture of samples from *multiple* datasets.

4.3.2 Active Learning System

Our aim is to employ self-supervised learning during active learning. The system is composed of an encoder (E) followed by a projector (P), and subsequently a classifier (C), as depicted in Figure 4.1. The overarching objective is to train an encoder to be invariant to artificially induced distortions, such as rotation, solarization, cropping, and more. The proposed model takes as input two distorted versions of an input vector, $x \in \mathbb{R}^D$, and produces a compressed latent vector, $z = f_\theta(x) \in \mathbb{R}^d$, where d is significantly smaller than D. Let $\mathcal{L}^0$ be the initial labeled pool, which constitutes a training set of data points in $\mathbb{R}^D$, representing the D-dimensional input space. Here, $x \in \mathbb{R}^D$ signifies a vector belonging to $\mathcal{X}$.

4.3.2.1 Barlow Twins

As stated in the introduction, we have employed Barlow Twins to extract low-level representations of our inputs. Here, we provide a more detailed explanation of the Barlow Twins

approach. A Barlow Twins network consists of an encoder (E) followed by a projector network (P), as illustrated in Figure 4.1, excluding the classifier component. To simplify the explanation, let's consider the scenario where the batch size is 1.

For each input image, two distorted versions, denoted as d^1 and d^2, are generated through different random data augmentations applied during training. These two distorted inputs are then passed through the encoder (E) and subsequently through a projector network (P), both equipped with trainable parameters. Consequently, the model produces two distinct low-level representations for the same input image, namely z^{d1} and z^{d2}, corresponding to the distorted versions d^1 and d^2.

It's worth noting that Barlow Twins employs a unique loss function that differentiates it from other self-supervised learning (SSL) methods, as elaborated below:

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{redundancy reduction term}} \tag{4.1}$$

Here, $\mathcal{C}$ represents the cross-correlation matrix computed between two identical networks, as defined below. Additionally, λ serves as a hyperparameter that defines the relative importance between the first and second terms of the loss.

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z_{b,i}^{d^1} z_{b,j}^{d^2}}{\sqrt{\sum_b \left(z_{b,i}^{d^1}\right)^2} \sqrt{\sum_b \left(z_{b,j}^{d^2}\right)^2}}$$

Here, b indexes batch samples, and i and j index the vector dimensions of the networks' outputs. The matrix C is square and has a size equal to the dimensionality of the network's output. Its values range from -1 to 1, where -1 indicates no correlation between $z_{b,i}^{d1}$ and $z_{b,i}^{d2}$, while 1 signifies a perfect correlation between $z_{b,i}^{d1}$ and $z_{b,i}^{d2}$.

4.3.2.2 Active Learning using Barlow Twins

With the aim of integrating both the self-supervised learning method Barlow Twins and active learning, we have introduced novel modifications to the existing Barlow Twins architecture, as illustrated in Figure 4.1, and elaborated further below. We have extended the current model, comprising an encoder (E) and a projector (P), by adding an additional classifier (C).

The entire system is trained using a modified and innovative loss function defined as follows:

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\log p_{\boldsymbol{\xi}}(\boldsymbol{y}|\boldsymbol{z})}_{\text{Classifer term}} + \gamma * (\underbrace{\sum_{i} (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \underbrace{\lambda \sum_{i} \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{redundancy reduction term}}) \tag{4.2}$$

Here, γ represents the weight assigned to the Barlow Twins loss, while the first term denotes the classifier loss. The optimization of the entire system is achieved by utilizing the joint loss, as demonstrated in Equation 4.2. Notably, it's important to highlight that the input to the classifier is the latent vector generated by directly passing the input image without distortions through the model (f).

4.3.3 Sampling technique

With the goal of selecting the b most informative data points from the unlabeled pool in conjunction with the trained model, let's delve deeper into the objective of the loss function outlined in Equation 4.2.

The primary aim of this loss function is to identify low-level representations that can capture as much information as possible about the inputs while minimizing their sensitivity to the applied distortions. To accomplish this, we have employed the Weibull sampling technique, initially introduced by Weibull [1951]. This technique serves the purpose of quantifying whether a sample can be considered an outlier or not. In our context, if a latent representation significantly differs from the latent representations in the labeled pool, it is classified as an outlier, as demonstrated by Gonzalez [1985] and utilized by Mandivarapu et al. [2020a].

The process begins by collecting all the latent vectors of images that have been correctly classified by the model at any stage of active learning. These latent vectors are then subdivided into respective clusters based on their class labels. Subsequently, the mean of each cluster is calculated, and the distances between the mean of each class and the remaining points are computed. We model the Weibull distribution using these distances for each class cluster.

Finally, when new unlabeled images are encountered, they are passed through the Weibull model to determine the extent to which each image is considered an outlier among all images. The top-ranked outlier images are then selected for labeling or annotation by the oracle.

Algorithm 3 Deep Active Learning using Barlow-Twins

Require: Unlabeled pool $\mathcal{U}^0$, labeled pool $\mathcal{L}^0$,number of labeling iterations t, initialize b (budget)
Require: Active Learning Model(f_θ), Optimizer
for $k = 1$ **to** t **do**
  Train f_θ on Labeled Pool ($\mathcal{L}^k$)
  Z $\leftarrow$ Collect the latent vectors of all correctly classified samples in Labeled Pool ($\mathcal{L}^t$)
  $\mathcal{C}_i \leftarrow$ Mapping of Z_i onto separate cluster per class $\mathcal{C}_i$
  $_i \leftarrow$ Calculate the distance of each point to its cluster $\mathcal{C}_i$
  $\mathcal{W}_i \leftarrow$ Map the distances by fitting them to a wiebull model
  Z $\leftarrow$ Collect the latent vectors of all the samples Unlabeled Pool ($\mathcal{U}^k$)
  for $Z = 1$ **to** n **do**
    Collect the outlier probability score using previously fitted wiebull model.
  Request labels for top b samples
  $L^{k+1} \leftarrow L^k \cup b$ samples.

  Train f on L^{k+1}.

4.4 Experimental Results

We performed experiments on four image classification datasets: MNIST [Deng 2012], CIFAR-10 [Krizhevsky 2009], and FashionMNIST [Xiao et al. 2017]—following the methodology defined in Sec. 4.3. Below, we first present our implementation details, then discuss our results.

4.4.1 Implementation Details

Hardware: We carried out our experiments on a Dell Precision 7920R server with two Intel Xeon Silver 4110 CPUs, two GeForce GTX 1080 Ti graphics cards, and 128 GBs of RAM.

Dataset sizes and budgets: As previously detailed in the methodology section, the term "budget" pertains to the number of samples labeled by the oracle during each round of active learning. The budget for each experiment is indicated in the legend accompanying the

respective results. The MNIST dataset comprises 50,000 images in its training set, which is further divided as follows: 100 images are allocated to the initial labeled pool, 5,000 images form a validation set, and the remaining 44,900 images constitute the unlabeled pool. Additionally, the MNIST dataset includes a test set comprising 10,000 images, which we employ to assess our model's performance after each stage of our active learning setup. In the experiments depicted in Figure 4.2, we utilized budgets of 100 and 1,000 samples. For the FashionMNIST dataset, we employed a similar setup. As for CIFAR-10, it bears similarities to MNIST in terms of the total number of images in the training and test sets. Within the CIFAR-10 training set, 5,000 images are designated for the initial labeled pool, 5,000 images are reserved for validation, and the remaining images are part of the unlabeled pool. In this case, we used a budget of 2,500 images per round of active learning, which amounts to 40% of the training data. The CIFAR-10 test set encompasses 10,000 images, serving as a benchmark to evaluate our model's performance after each stage of the active learning setup.

Runs: For all the experiments, we measured performance by computing the average accuracy across 5 independent runs.

State of the art comparison: We compared our method against several recent AL approaches including Variational Adversarial Active Learning (VAAL) [Sinha et al. 2019], Core-Set [Sener and Savarese 2017], Monte-Carlo Dropout [Gal and Ghahramani 2016], Ensembles using Variation Ratios (Ensembles w. VarR) [Freeman 1965] [Beluch et al. 2018], Deep Bayesian AL (DBAL) [Gal et al. 2017], BatchBALD [Kirsch et al. 2019], and WAAL([Shui

et al. 2020]). As a baseline, we also included uniform random sampling (Random) since it remains a competitive strategy in the field of active learning.

Architectures: For experiments on MNIST and Fashion-MNIST we used a LeNET network [Lecun et al. 1998] as the encoder, a projector network, followed by a classifier. We used latent vectors of size 60. As noted in Sec. 2.2, the classifier consists of a single linear layer. For CIFAR-10, we used a VGG16 network [Simonyan and Zisserman 2014] as our encoder and a latent vector of size 512 followed by classifier with single layer.

Optimization: We optimized the entire system using a mini-batch size of 64, a learning rate set to 0.001, a Barlow Twins constant of 0.001, and a weight decay of 10^{-5}. Our optimization process spanned 150 epochs at each stage of training. Upon completing a stage, we employed the Weibull sampling method to request labels for b images from the unlabeled pool. Once we received the labels for these images from the oracle, they were incorporated into the labeled pool and utilized in the subsequent stages of learning.

Image Augmentations

We employ augmentations similar to those used in Barlow Twins and other SSL approaches. As illustrated in Figure 4.1, two distorted images are generated from a single input image by applying various transformations. The image augmentation pipeline initiates with random cropping and resizing, which are applied to all images. Subsequently, it includes Gaussian blurring, color jittering, conversion to grayscale, horizontal flipping, and solarization. Notably, the last five augmentations are applied randomly during the process.

Computer Vision Task results: To evaluate the effectiveness of our method we tested

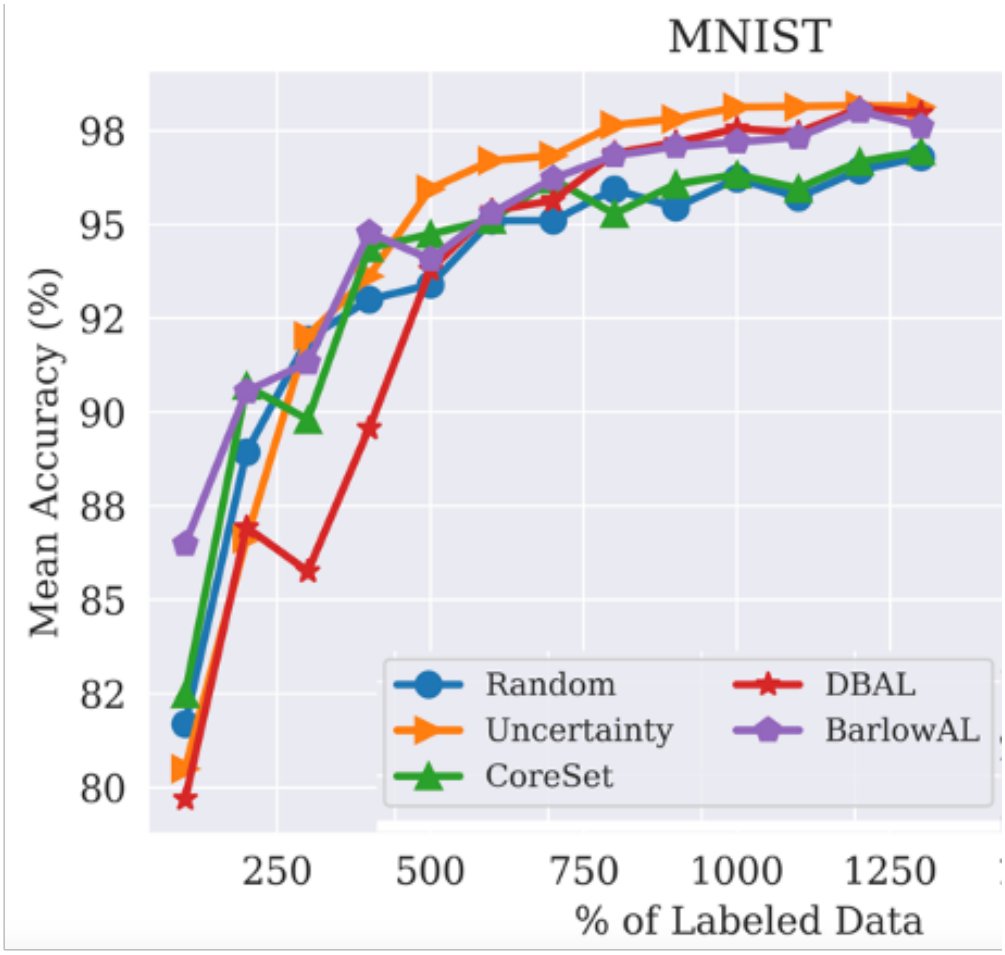


Figure 4.2 Robustness of our approach on MNIST classification . Our results confirm that our approach significantly outperforms this baseline.

our method on MNIST,CIFAR-10, Fashion MNSIT and mixture of multiple datasets in the unlabeled pool.

MNIST: We conducted experiments on the MNIST dataset with an initial labeled pool size of 100 and a budget size of 100 at each stage of active learning. As illustrated in Figure 4.2, our method performed on par with the existing methods. Given that this is a relatively straightforward computer vision task, all methods performed within a similar range.

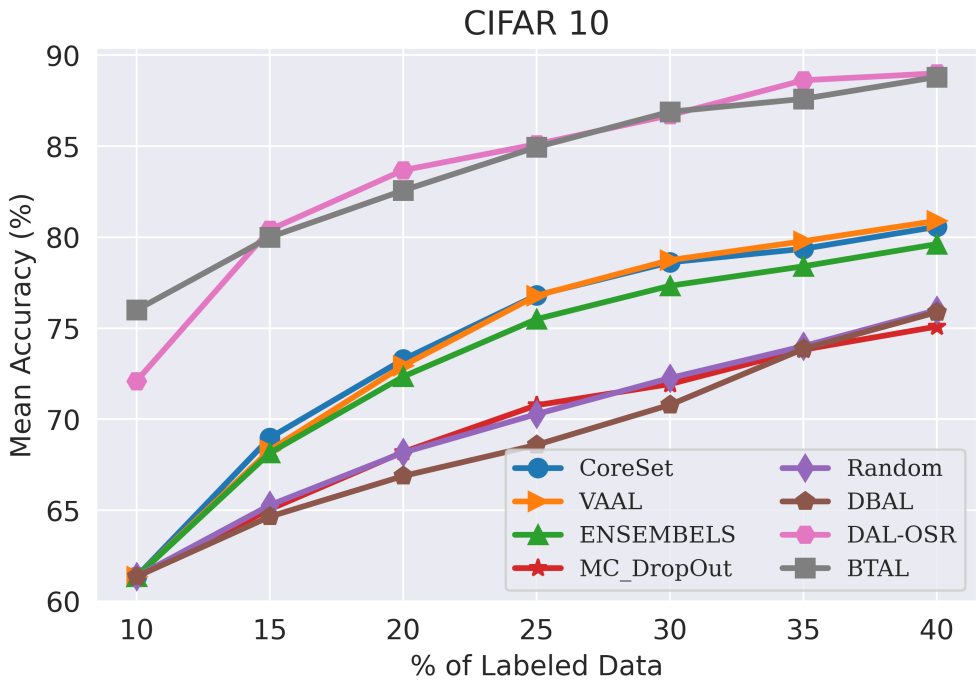


Figure 4.3 Robustness of our approach on CIFAR10 classification . Our results confirm that our approach significantly outperforms this baseline. Note, our approach is marked as Barlow-Twins Active Learning (BTAL).

CIFAR-10:

We conducted experiments on the MNIST dataset with an initial labeled pool size of 100 and a budget size of 100 at every stage of active learning. As depicted in Figure 4.2, our method performed comparably to existing methods. Given that this is a relatively straightforward computer vision task, all methods performed within a similar range.

Moving on to CIFAR-10, we conducted two separate experiments with different budget sizes. In the first experiment, the initial labeled pool consisted of 5,000 images, and the budget (b) was set to 2,500 at each stage. As shown in Figure 4.3, our proposed method performed on par with the existing state-of-the-art method DAL-OSR. VAAL ranked third

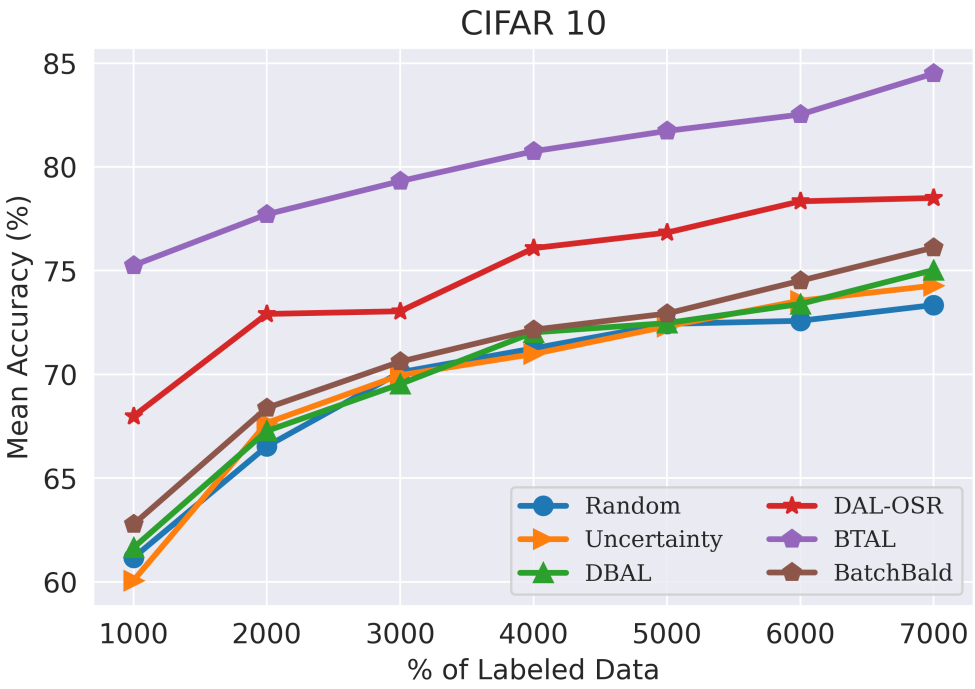


Figure 4.4 Robustness of our approach on CIFAR10 classification . Our results confirm that our approach significantly outperforms this baseline. Note, our approach is marked as Barlow-Twins Active Learning (BTAL).

with an accuracy of 80.71%, followed by Core-Set with an accuracy of 80.37%, and Ensemble with VarR at 79.465%. Random sampling, DBAL, and MC-Dropout lagged significantly behind other methods.

To assess the effectiveness of the proposed model compared to other methods with small budgets, we designed experiments where the initial labeled pool contained 5,000 images, and the budget (b) was limited to 1,000 at each stage of active learning. As shown in Figure 4.4, the proposed method outperformed all existing methods by a substantial margin, with DAL-OSR ranking second, followed by BatchBALD, while the remaining methods performed within a similar range. This demonstrates the efficacy of the proposed method, especially

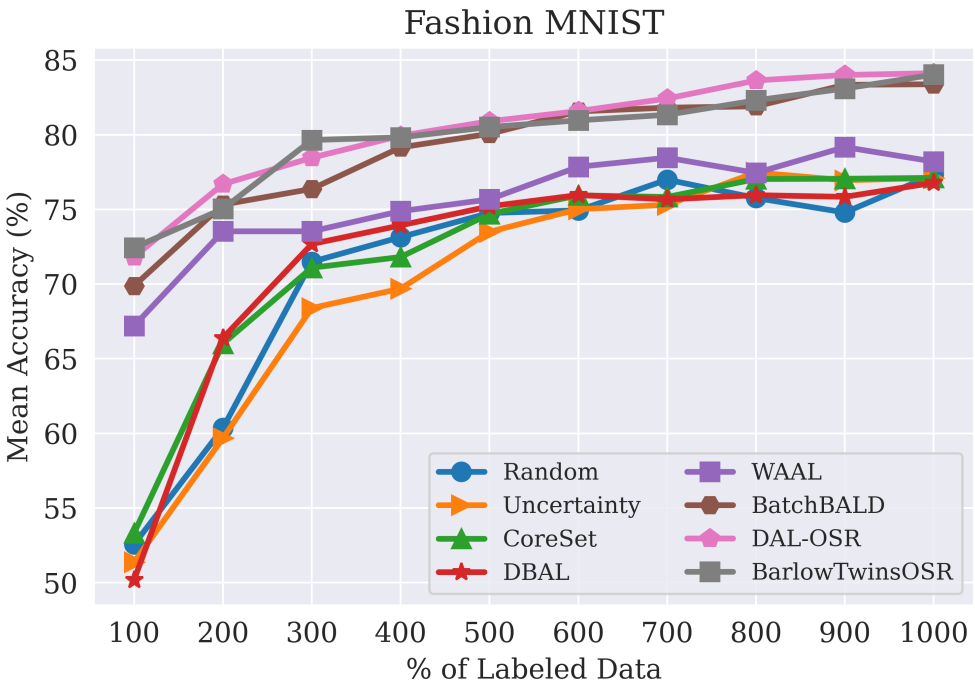


Figure 4.5 Robustness of our approach on FashionMNIST classification . Our results confirm that our approach significantly outperforms this baseline.

when the budget size is relatively low. The original accuracy achievable using the entire CIFAR-10 dataset was 92.63%.

FashionMNIST:

To assess the robustness of our approach across different datasets, we conducted experiments on another standard benchmark dataset, FashionMNIST. Similar to our previous experiments, we compared our method with other existing state-of-the-art methods such as DAL-OSR, Core-Set [Sener and Savarese 2017], Deep Bayesian AL (DBAL) [Gal et al. 2017], BatchBALD [Kirsch et al. 2019], and WAAL [Shui et al. 2020].

As demonstrated in Figure 4.5, our method outperforms all other methods and achieves performance on par with DAL-OSR.

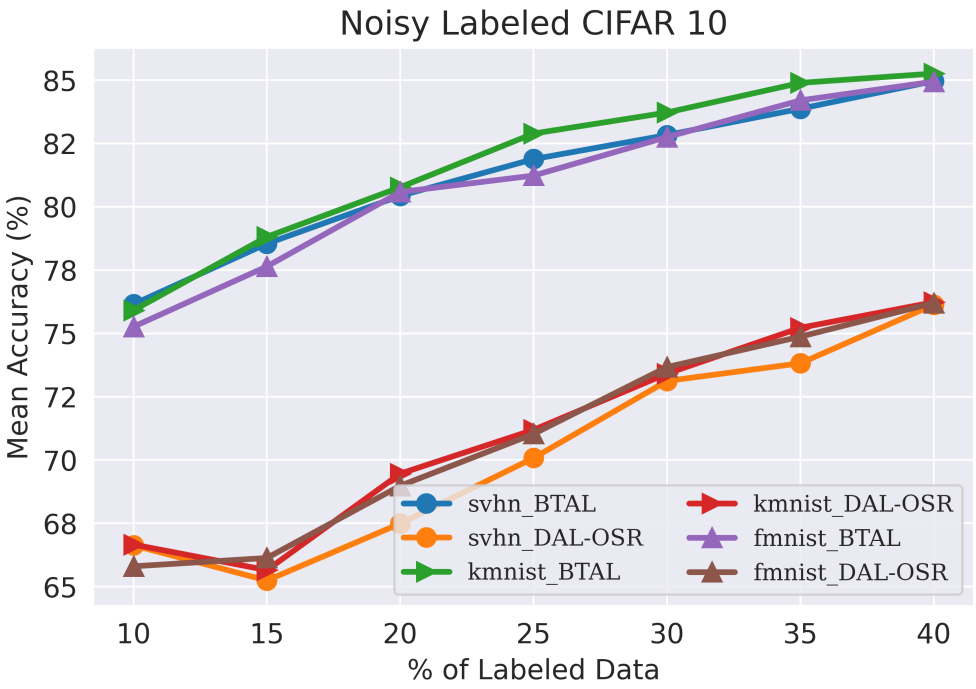


Figure 4.6 Robustness of our approach on CIFAR10 classification tasks when the unlabeled pool includes samples from either the SVHN, KMNIST, or FashionMNIST datasets. The first three curves used the M_2 classifier, while the ones with the 'Random' subscript used random sampling. Our results confirm that our approach significantly outperforms this baseline.

Mixed UnLabeled Pool: Lastly, we conducted tests to assess the extreme case of active learning, as proposed in DAL-OSR. We adopted a similar setup to DAL-OSR, where 10,000 images from other datasets such as SVHN, KMNIST, and KMNIST were mixed into the source dataset of CIFAR-10. Consequently, the proposed method needed to distinguish not only between informative and non-informative samples but also between in-distribution data samples (CIFAR-10) and out-of-distribution samples (SVHN, KMNIST, KMNIST).

A superior model effectively allocates the budget, selecting informative in-dataset samples. For example, when the budget is set at 1,000, if the model chooses 1,000 samples, out of which 400 belong to the out-of-distribution dataset, only 600 samples are sent for

annotation and added to the labeled pool dataset. The remaining 400 samples are returned to the unlabeled pool, resulting in a total of 10,000 out-of-distribution data samples in the unlabeled pool at every stage of active learning. As active learning progresses through stages, it becomes increasingly challenging for the model to select in-label samples, given the diminishing proportion of in-label samples and the increasing presence of out-of-distribution samples in the unlabeled pool.

4.5 Conclusions and Future work

In this study, we introduced a novel approach for active learning in the presence of both iid and non-iid shifts within unlabeled pools for computer vision tasks. Our method combines self-supervised learning techniques with the Weibull sampling method. We conducted a comprehensive evaluation of our approach through comparisons with existing methods using three open-source datasets. We conducted rigorous benchmarks against state-of-the-art active learning models in computer vision tasks. Additionally, we presented studies involving different budget allocations and mixed unlabeled pool setups to demonstrate the effectiveness of our method compared to other approaches. The results consistently showed that our method performed at par or even outperformed existing baselines in computer vision tasks. For future work, we aim to further explore more effective self-supervised methods to handle active learning at scale.

CHAPTER 5

Continual Learning with Deep Artificial Neurons

5.1 Introduction

Humans and other organisms are capable of acquiring new knowledge over time without completely forgetting the past, an ability commonly referred to as continual learning or learning without forgetting [Beaulieu et al. 2020; Flennerhag et al. 2020; Javed and White 2019; Kirkpatrick et al. 2016; Li and Hoiem 2016]. Unfortunately, this ability has largely eluded artificial learning systems. Deep neural networks, in particular, are prone to catastrophic forgetting because backpropagation updates all the weights in the network during learning, which causes old features to be overwritten by new ones. Despite this drawback, backpropagation is a powerful technique because it directly correlates weight updates to their impact on the loss function. As noted by [Guerguiev et al. 2019], "any learning system that makes small changes to its parameters will only improve if the changes are correlated to the gradient of the loss function." This insight suggests that any architectural or algorithmic change to enable continual learning must (1) either be compatible with backpropagation or (2) offer an alternative way of assessing the impact of a given weight on performance.

In this paper, we propose Deep Artificial Neurons (DANs), a novel, backpropagation-compatible neural architecture that facilitates continual learning. As we detail below, networks with DAN units can learn over time with minimal forgetting using standard backpropagation and without the need for an experience replay buffer or sleep/wake cycles. Our DAN architecture is inspired by biological neurons, which, unlike their simple artificial coun-

terparts, are extraordinarily complex [Beniaguev et al. 2020; Guergiuev et al. 2017; Hawkins and Blakeslee 2004; Izhikevich 2007; Jones and Kording 2020; Kandel et al. 2000; Palavalli et al. 2020]. In particular, the strength of connections between biological neurons is best modeled as a *vector-valued* function between the pre- and post-synaptic neurons. This vector represents the state of chemical concentrations, electrical potentials, proximity, and other temporal and spatial dynamics that affect how strongly a cell responds to its inputs. As we detail below, DANs perform a similar function in our networks; namely, they filter the output signals of neurons in one layer before passing it on to the subsequent layer. As our experiments show, this non-linear, vector-valued filtering allows a network with DAN units to learn over time, using only standard backpropagation, with minimal forgetting.

5.2 Related Work: Neurons, Meta-Learning, Continual Learning

Many have begun to acknowledge that efficient learning in real brains results from innate priors which have arisen through evolution, and which are generally kept fixed during intra-life deployment [Zador 2019]. It is unsurprising, therefore, that there have been some very recent attempts to embed more powerful priors in artificial neural networks [Aguera y Arcas 2302; Beniaguev et al. 2020; Gregor 2020; Guergiuev et al. 2017; Hawkins and Ahmad 2016; Jones and Kording 2020; Trabelsi et al. 2017]. In a highly original work, [Mordvintsev et al. 2020] showed that it may be beneficial to think of individual cells, themselves, as networks with self-organizing properties. However, little work has been done to investigate whether this increase in computational power at the cellular level might improve performance

specifically in continual learning settings.

Machine learning algorithms generally rely upon the *iid* assumption, which asserts that the training samples are independent of each other (i.e., no correlation between successive samples), and that the training and test distributions need to be (approximately) the same. For this reason, the most reliable technique to date for overcoming catastrophic forgetting (CF) given non-*iid* data is one which explicitly enforces this assumption, often referred to as experience replay (ER) [Mnih et al. 2013]. A severe limitation of ER, however, is that it requires a system to retain all (or most) of the data that it encounters. This necessitates additional hardware, increases training time, and can even worsen learning efficiency. Therefore, three categories of alternative techniques have emerged to combat CF in settings where all data cannot be retained: (1) regularization-based approaches [Farajtabar et al. 2019; Kirkpatrick et al. 2016; Lesort et al. 2019], (2) knowledge-compression, or capacity expansion [Joseph and Balasubramanian 2020; Mandivarapu et al. 2020b; von Oswald et al. 2019b; Yoon et al. 2018], and (3) meta-learned representations and update rules [Beaulieu et al. 2020; Flennerhag et al. 2020; Gregor 2020; Javed and White 2019; Lindsey and Litwin-Kumar 2020].

For this paper, we draw strongest inspiration from a promising approach known as Warped Gradient Descent (WGD) [Flennerhag et al. 2020]. WGD mitigates catastrophic forgetting by meta-learning warp parameters ω, realized as warp-*layers*, which are interleaved between the standard layers of a neural network. These warp parameters are held *fixed* during deployment, allowing the rest of the network to learn, without forgetting, using standard

backpropagation. As the name implies, parameters ω warp activations in the forward pass, and the gradients in the backwards pass. However, in contrast to WGD, which uses dense fully connected warp layers, we show that it is possible to learn small, common networks φ, constituting shared neuronal phenotypes (DANs) that can be distributed throughout larger networks of plastic Synapses, thereby influencing their learning trajectories.

5.3 Problem Formulation

In this work, we propose to meta-learn a single parameter vector φ, shared by all DANs in the model, which can mitigate catastrophic forgetting in a network that *learns during deployment* by updating its Synapses with standard backpropagation. We call this parameter vector a neuronal phenotype, since it defines the behavior of each DAN in the network, and it is kept fixed during intra-lifetime deployment. Specifically, we consider Continual Learning Trajectories (e.g. $[\mathcal{T}_0, \mathcal{T}_1, ..., \mathcal{T}_k]$) which are comprised of sequences of tasks $\mathcal{T}$. These task sequences are drawn uniformly from some underlying task distribution $p(\mathcal{T})$. We add a single-context stipulation stating that within a sequence of tasks, each data sample $x \in \mathcal{T}_i$ is mapped to one and only one target value y, and each sample x belongs to one and only one task $\mathcal{T}_i$. In our experiments, we assume that tasks are disjoint, and therefore our model must learn one task before moving on to the next, though we anticipate that we can relax this assumption to handle overlapping or evolving task distributions in future work. The model is therefore allowed to perform a fixed number of updates on data from each task, where tasks are encountered one after the other. We seek a model which retains good performance over

the whole task trajectory, without being allowed to revisit data from previously encountered tasks in the sequence.

We formulated an experiment similar to the one proposed in Flennerhag et al. [2020], and originally in Finn et al. [2017]. More specifically, we consider the problem of sequential non-linear regression, wherein a model must try to fit to a complete function, when exposed to data from only part of that function in distinct time-intervals. In other words, it must learn the complete function in a *piece-wise*, or incremental manner, since it cannot revisit data to which it was exposed during previous intervals. As in Flennerhag et al. [2020], we split the input domain $[-5, 5] \subset \mathbb{R}$ into 5 consecutive sub-intervals, which correspond to 5 distinct tasks. Task_1 therefore corresponds to the sub-function falling within $[-5, -3)$; Task_2 corresponds to the sub-function within $[-3, -1)$, and so on. The model is exposed to Tasks 1 through 5 in sequential manner. During each sub-task, the network is exposed to 100 data points, drawn uniformly from the current task window. That is, during Task_1 the model performs 100 updates on data sampled from $[-5, -3)$. In our experiments, we perform 1 update on every sample, equating to a batch size of 1. Sub-tasks are thus defined by their respective windows in the input domain.

We slightly modify the target functions used in Flennerhag et al. [2020]. We define a task sequence by a target function that is a mixture of two sine functions with varying amplitudes, phases, and x-offsets. At the beginning of each meta-epoch, we randomly sample two amplitudes $\alpha_{(0,1)} \in (0,2)$, phases $\rho_{(0,1)} \in (0, \pi/3)$, and x-offsets $\phi_{(0,1)} \in [-5, 5]$. Summing two such sine functions yields a target function of the form:

$$y = \alpha_0 sin((\rho_0 x) + \phi_0) + \alpha_1 sin((\rho_1 x) + \phi_1) \tag{5.1}$$

Afterwards, we discard any target functions from the training distribution meeting the following criteria: $y_{max} > .8$; $y_{min} > -.8$; $y_{max} - y_{min} < .4$. This yields a final input domain of $[-5, 5]$ and output range of $[-.8, .8]$.

5.4 Model Architecture

Deep Artificial Neurons, or DANs, are themselves realized as multi-layer neural networks. For the purposes of demonstration, consider a DAN instantiated as a 2-layer neural network, with a single layer of hidden nodes, and a single output node (i.e. the output activation of the neuron). In practice, we apply a *tanh* non-linear activation to the output of the hidden layer, as well as to the output layer of each DAN. See the bottom of Fig. 5.3 for an illustration. Let n_channels denote both (1) the size of the input vector to this network, and (2) as we will see, the number of connections between pairs of DANs. Conceptually, we can distribute this single DAN amongst all nodes of a traditional neural network. Fig. 5.1 offers an illustration of how to convert a standard ANN into a network of DANs with n_channels=3.

More generally, consider the topology of a standard, fully-connected, feed-forward neural network with l layers of nodes, and let n_l denote the number of nodes in layer l. Let l_0 be a special case, denoting the layer of input nodes, which are *not* DANs. We can convert this topology to a network of DANs in the following way. For each layer of nodes, up to but

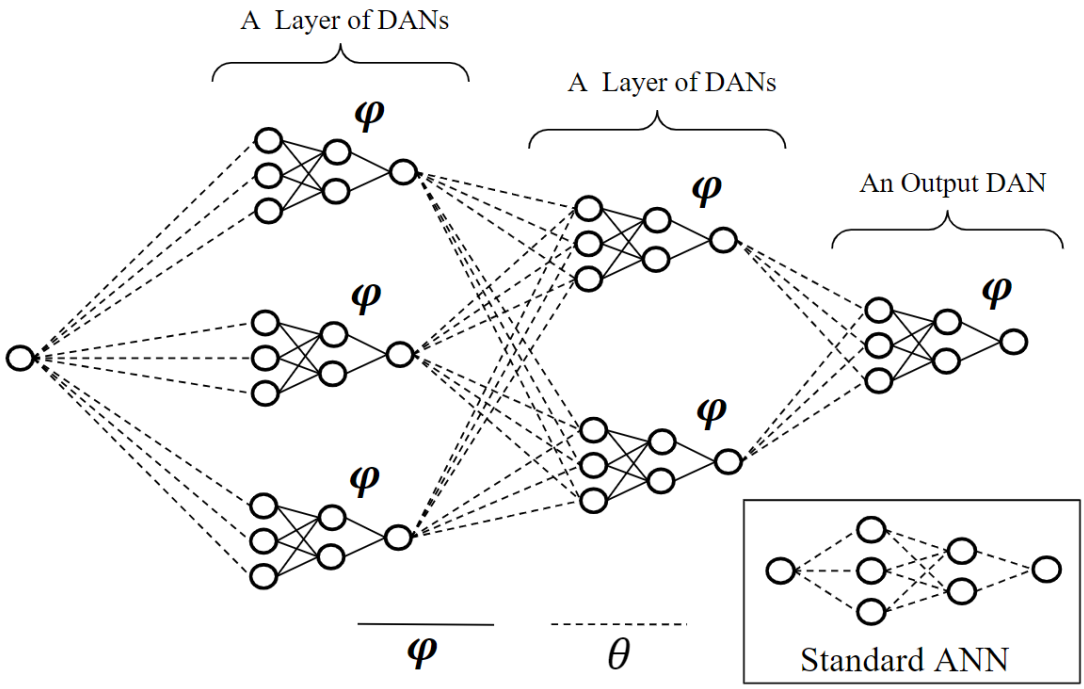


Figure 5.1 A Network of Deep Artificial Neurons (DANs). DANs are connected to one another by parameters θ, which can be regarded as Vectorized Synapses, or VECs. All DANs may share parameters φ, which we dub a neuronal phenotype. Synapses in a Network of DANs are vectors. The strength of the connection between 2 DANs is therefore an non-linear function of the magnitude and orientation of this synaptic vector.

not including the layer of output nodes, we instantiate a layer of Synapses as a standard, fully-connected weight-matrix θ_l with dimensions $n_l \times (n_{l+1} \times n\_channels)$. Synapses connect layers of DANs to one another. Feed-forward propagation of a signal along these connections is therefore facilitated in the standard way, by computing the dot product of the activation vector σ_{l_out} from the previous layer and this layer of Synapses θ_l. This yields a large input vector $\sigma_{(l+1)\_in}$, to be processed by the DANs in the next layer:

$$\sigma_{(l+1)\_in} = \sigma_{l\_out} \cdot \theta_l = \sum_{i}^{n_{l+1}} \theta_i^j \sigma_{l\_out}^j + b_i, \tag{5.2}$$

where j denotes the index of nodes in layer l, and i is the index of nodes in layer $l+1$. Note that in practice we do *not* apply a non-linear activation function to this vector. Rather, n slices of the raw dot product $\sigma_{(l+1)\_in}$ are fed as input to the n DANs in the next layer. That is, since all DANs share parameters φ, the same DAN model processes each slice of $\sigma_{(l+1)\_in}$. Said another way, the input vector to the layer of DANs is sliced into n_{l+1} equally sized sub-vectors, where n is the number of DANs in layer $l+1$. The output vector of a layer of DANs is obtained by passing each of these separate slices through the DAN, and concatenating the resulting activations. This can be done very efficiently by simply reshaping the inbound synaptic activations $\sigma_{(l+1)\_in}$ and using our single DAN phenotype to process all slices as a batch. This results results in $\sigma_{(l+1)\_out}$ which constitute the output activations of the DANs in layer $l+1$.

In practice, we also use skip connections, inspired by Deep Residual Networks He et al. [2015] and Flennerhag et al. [2020], in order to facilitate efficient learning. Skip connections are realized as additional layers of Synapses $\theta_{skip(j,k)}$, with dimensions $n_j \times (n_k \times n\_channels)$, which bypass layers of DANs by providing a direct pathway from nodes in layer j to layer k, where $k = j+2$. This allows some information to be sent directly downstream, without being subjected to processing by the intermediate layer of DANs. When using skip conenctions, the input vector to a layer of DANs in layer k is obtained by summing the vector computed in Equation 5.2 with the vector signal traveling along $\theta_{skip(j,k)}$:

$$\sigma'_{k\_in} = \sigma_{k\_in} + (\sigma_{j\_out} \cdot \theta_{skip(j,k)}) \tag{5.3}$$

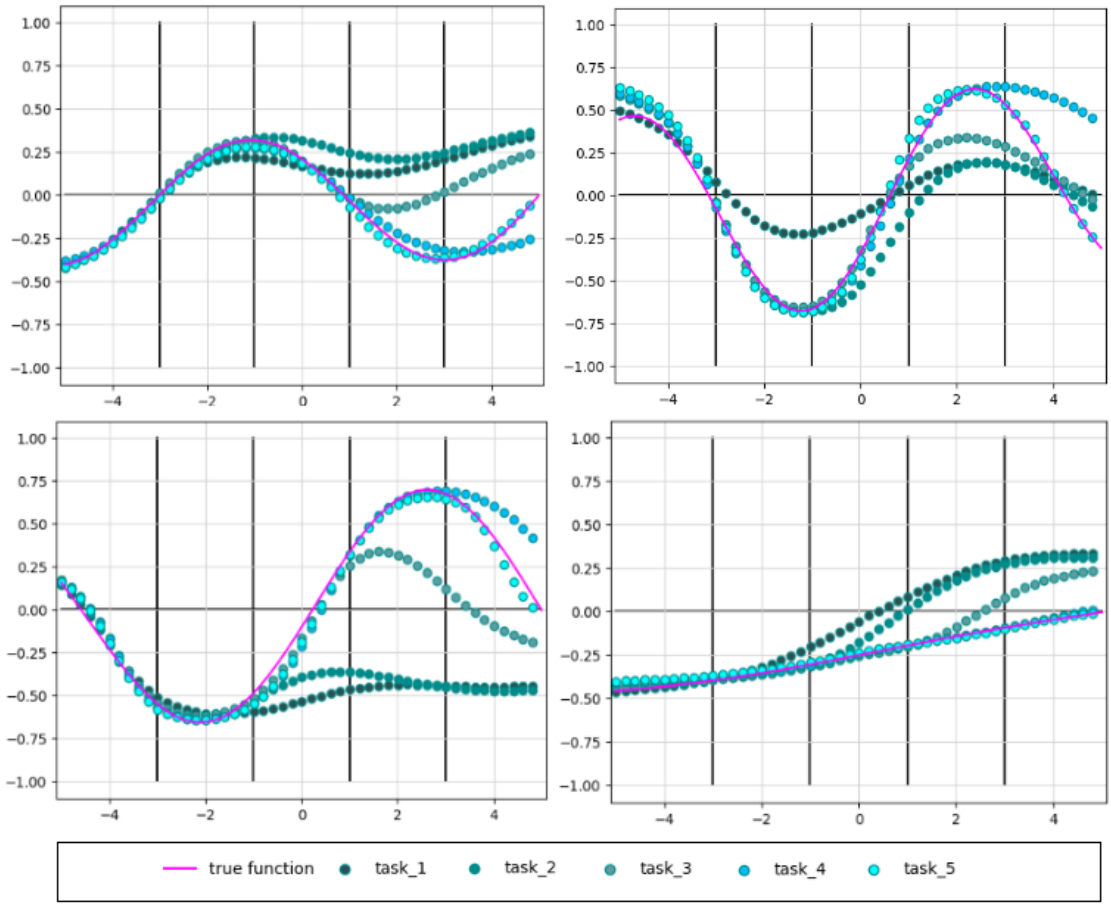


Figure 5.2 Continual Learning during Deployment on 4 non-linear functions, each divided into 5 sub-tasks, using a meta-learned neuronal phenotype which is held fixed. Synapses are updated with standard Backpropagation. In each of the 4 plots, each color in the plot depicts the predictions of the model over the whole function after performing 100 updates on data from the current sub-task only. In other words, the darkest plot represents the model's predictions over the whole function after training only on task_1 $[-5, -3)$. During the next stage of learning, the model performs 100 updates on data from task_2 only $[-3, -1)$. The lightest plot (cyan) depicts the model's predictions after the last round of learning: 100 updates on data from task_5 $[3, 5]$. As shown, the model retains a good fit over the whole function even when it learns these sub-tasks in a sequential manner.

The result is a complete model, comprised of 2 distinct sets of parameters: Synapses, parameterized by θ, and DANs parameterized by φ. As we will show in coming sections, Synapses are intended to be fully plastic at all times. The parameters of our DANs, our

so-called neuronal *phenotype*, are meta-learned and then held fixed during deployment.

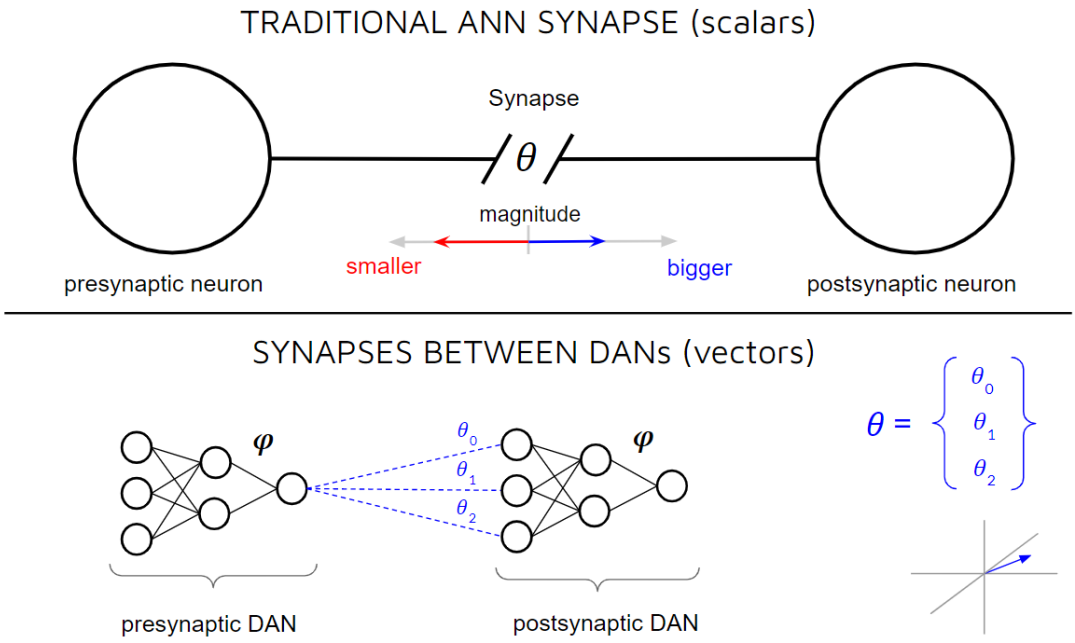


Figure 5.3 In traditional artificial networks, synapses are represented as single scalar values. In contrast, within a network of DANs, synapses are n-dimensional vectors. This distinction allows the strength between two DANs to be expressed as a non-linear function, dependent on both the magnitude and orientation of the synaptic vector.

As in Flennerhag et al. [2020], we leverage the benefits of a unique set of parameters φ which can be shown to warp the gradients applied to another set of parameters θ in order to prevent catastrophic forgetting. In contrast to WGD Flennerhag et al. [2020], however, we show that a single, small network, parameterized by φ, is sufficient to facilitate our meta-objective, rather than unique, separate layers of warp parameters. Additionally, we feel that our approach offers an additional layer of biological plausibility, and might help to explain some of the behavior and responsibilities of real neurons.

5.5 Methodology

Meta-Learning is generally concerned with optimizing some meta-objective over a distribution of tasks in order to attain some innate proficiency at comparable tasks likely to be encountered during a separate, deployment phase. To accomplish this, most meta-learning algorithms employ an inner-loop/out-loop framework, wherein optimization over several specific tasks occurs in the inner-loop, and proficiency at the meta-objective is evaluated and optimized in the outer-loop. Inspired by Warped Gradient Descent (WGD) Flennerhag et al. [2020], we adopt such an approach, and wish to meta-learn parameters which facilitate continual learning during deployment.

During meta-training, we randomly sample target functions of the form defined in Equation 5.1. These target functions are split into 5 sub-tasks, as explained in Section 5.3. We deploy our model on each target function, and sub-tasks are encountered sequentially. Optimization over a single sub-task is done by performing backpropagation on both sets of parameters, θ and φ, using the loss over the current sub-task. This is known as an inner-loop epoch. At the end of each inner-loop epoch, we quantify the Meta-Loss over subtasks [0,...,*cur*], where *cur* is the current sub-task. Meta-Optimization is done by performing backpropagation on parameters φ ***only***, using the Meta-Loss. Repetition of this process over a sequence of 5 sub-tasks, given the current target function, is known as a meta, or outer-loop epoch. At the end of each outer-loop epoch, we sample a new target function, and repeat the process.

More formally, let the Historical Learning Trajectory $\mathcal{H}_t$ represent the dataset [x_0, x_1,

$\ldots, x_t]$ comprised of all data encountered by the system, prior to and including timestep t. Note that $\mathcal{H}_t$ therefore contains data from one or more sub-tasks $\mathcal{T}$. We have Synapses, parameterized by θ, and DANs, parameterized by φ, which together define the complete Model.

Note that since DANs are distributed throughout the network, the gradients for Synapses ∇_θ depend on parameters φ, and that the meta-gradient ∇_φ depends on parameters θ. This is true, since each set of parameters θ and φ are factors of both gradients.

We can define a Model State at timestep t as $\theta_t\varphi_t$. Given a new sample at timestep $t+1$, this Model State will result in a measurable Task Loss $\mathcal{L}_\mathcal{T}$, for which we can compute a gradient $\nabla_{\theta_t\varphi_t}\mathcal{L}_\mathcal{T}^{t+1}$. Note that we can factor this gradient into its distinct components:

$$\nabla_{\theta_t\varphi_t}\mathcal{L}_\mathcal{T}^{t+1} = \nabla_{\theta_t}\nabla_{\varphi_t}\mathcal{L}_\mathcal{T}^{t+1} \tag{5.4}$$

This is desirable since we may want to assign separate learning rates to each set of parameters. For instance, let α denote the learning rate for parameters θ, and let γ denote the learning rate for parameters φ. Since we update both θ and φ in the inner loop, when learning individual sub-tasks, performing an inner-loop update on data from the current task at timestep $t+1$, like so:

$$\theta_{t+1}\varphi_{t+1} \leftarrow \theta_t\varphi_t - \alpha\nabla_{\theta_t}\gamma\nabla_{\varphi_t}\mathcal{L}_\mathcal{T}^{t+1} \tag{5.5}$$

...results in the new Model State $\theta_{t+1}\varphi_{t+1}$. Note that this update, $\theta_{t+1}\varphi_{t+1} \leftarrow \theta_t\varphi_t$, may

have caused forgetting over $\mathcal{H}_{t+1}$, which now includes the latest data sample x_{t+1}.

We can quantify the Memory Loss over $\mathcal{H}_{t+1}$, defined as $\mathcal{L}_{\mathcal{M}}^{\mathcal{H}_{t+1}}$. This constitutes the meta-loss, which we wish to minimize in order to *facilitate* our meta-objective during inner-loop deployment.

Specifically, we seek an optimal neuronal phenotype, defined by a single parameter vector φ^*, shared by all DANs, which would have resulted in the least amount of forgetting over $\mathcal{H}_{t+1}$. Said another way, had the original state of the model been $\theta_t\varphi_t^*$, instead of $\theta_t\varphi_t$, then the inner loop update would have been:

$$\theta_{t+1}^*\varphi_{t+1}^* \leftarrow \theta_t\varphi_t^* - \alpha\nabla_{\theta_t}\gamma\nabla_{\varphi_t^*}\mathcal{L}_{\mathcal{T}}^{t+1} \tag{5.6}$$

This would have resulted in an alternative Model State $\theta_{t+1}^*\varphi_{t+1}^*$, ideally resulting in less forgetting than that originally induced by $\theta_t\varphi_t$.

Therefore, we calculate the Memory Loss across $\mathcal{H}_{t+1}$, using the current model state $\theta_{t+1}\varphi_{t+1}$, and compute the gradient w.r.t. this quantity. By taking a step towards φ_t^*, we update the phenotype φ, and in the process attempt to minimize the meta-loss. We can do this by factoring the gradient and isolating the update to φ only:

$$\varphi'_{t+1} = \varphi_{t+1} - \gamma\nabla_{\varphi_{t+1}}\mathcal{L}_{\mathcal{M}}^{\mathcal{H}_{t+1}} \quad \text{s.t.} \qquad \varphi'_{t+1} \approx \varphi_t^* \tag{5.7}$$

The full meta-training procedure is outlined in Algorithm 5.5.

After meta-training is completed, the model is *deployed*. DAN parameters φ are held

Meta-Learning a Neuronal Phenotype for Continual Learning

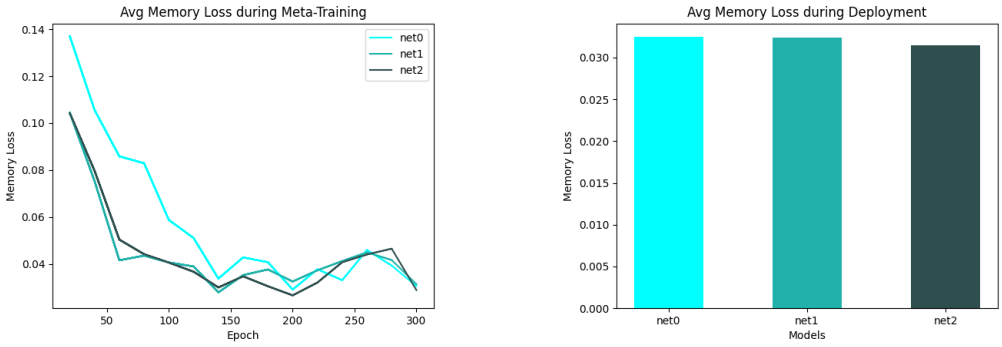
Require: $p(\mathcal{T})$: distribution over target functions
Require: α, γ: learning rate hyperparameters
Require: inner_steps: number of inner loop steps
$\theta \leftarrow \theta_0, \varphi \leftarrow \varphi_0$: randomly initialize the model
while not done **do**
  Sample a Target Function $\mathcal{T} \sim p(\mathcal{T})$
  for sub-task st in $\mathcal{T}$ **do**
    for t in inner_steps **do**
      Perform an update: $\theta_{t+1}\varphi_{t+1} \leftarrow \theta_t\varphi_t - \alpha\nabla_{\theta_t}\gamma\nabla_{\varphi_t}\mathcal{L}^t_{st}$
    end for
    $\mathcal{H}_{t+1} \leftarrow$ data from sub_tasks[0,...,st] $\subset \mathcal{T}$
    Compute Memory-Loss over $\mathcal{H}_{t+1}$
    Update Phenotype: $\varphi'_{t+1} = \varphi_{t+1} - \gamma\nabla_{\varphi_{t+1}}\mathcal{L}^{\mathcal{H}_{t+1}}_{\mathcal{M}}$
  end for
  $\theta \leftarrow \theta_0$: reset VECs to initialization
end while

fixed, and the model is obligated to learn continually, without forgetting, using standard backpropagation. That is, θ update normally, while DANs remain fixed. We offer empirical validation of our approach in the next section.

5.6 Experiments and Results

For all experiments, we used a network topology of 1 input node, 2 hidden layers of 40 nodes each, and a single output node. Recall that, apart from the single node in the input layer, each node represents a DAN, and the topology is therefore *converted* to a network of DANs. To this topology, we added 2 skip layers, as described in Section 5.4: from layer 0 to layer 2, and also from layer 1 to layer 3. The DAN itself is a 3 layer neural network with $n\_channels$ input nodes, followed by a hidden layer with 15 nodes, another hidden layer with 8 nodes, and a single output node, parameterized by φ. We applied *tanh* activations to the hidden and output layers of the DAN. For all experiments except that depicted in

Fig.5.4, we set $n\_channels = 40$. For Meta-Training, we set the learning rate for Synaptic parameters $\theta = .001$, and the learning rate for DAN parameters $\varphi = .0001$.



(a) (b)

Figure 5.4 Model definitions: **net0** uses a single, shared phenotype (cell-type); **net1** uses one phenotype for each layer; **net2** does not enforce parameter sharing among any DANs. (left) Minimization of the Memory-Loss during Meta-Training. (right) Avg Memory Loss during Deployment is nearly equal for all models, and nearly identical to the loss achieved near the end of meta-training, $\approx .03$ (mean squared error) across the full task-trajectory after learning 5 sub-tasks in sequence.

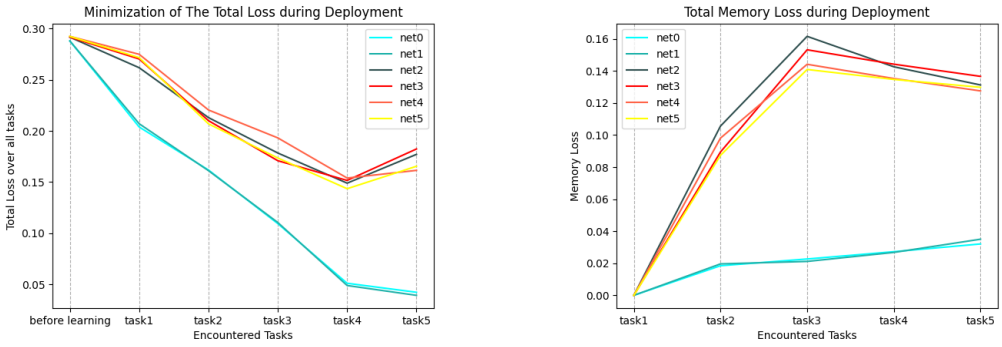
Fig.5.2 shows the ability of a meta-trained model to learn continually during ***deployment***; when it encounters tasks in a sequential manner, and is obligated to retain a good fit over previous sub-tasks, even though it is exposed to data from each task only once. During this experiment, DAN parameters φ were held fixed, and the network learns by using standard backpropagation to update Synapses θ.

Fig.5.5 depicts minimization of the Memory-Loss, our meta-objective, during meta-training. We found that the model converges relatively quickly, requiring only 200-300 meta-epochs to find a suitable phenotype, though this is likely due to task simplicity. Additionally, as the Figure shows, we sought to isolate the effect of using a single set of parameters for DANs in the whole network. To do this, we compared 3 models: one which used a single

parameter vector for all DANs (net0: a single phenotype throughout the network), another which used a separate parameter vector for each *layer* of DANs (net1: phenotypes unique to each layer), and a third which did not enforce any parameter sharing amongst DANs (net2; 81 unique DANs in the network).

In above fig we compare the abilities of various models during deployment, when they are confronted with tasks in a sequential manner, and obligated to learn continually. Specifically, we sought to verify whether the meta-learning procedure was indeed endowing the DANs with an innate ability to assist in learning without forgetting. These plots confirm that hypothesis, showing that a meta-learned phenotype outperforms random parameter vectors, regardless of whether they are fully plastic during deployment, or fixed.

Finally, we investigated the effect of the size of $n\_channels$ on the ability of the model to minimize Memory-Loss during meta-training. Specifically, we asked, is there indeed a benefit to vectorizing the connections between pairs of DANs, and in the process increasing the size of the input to each DAN? Above figure shows that the answer to that question was also yes. The plot shows that as the number of (1) connections between pairs of neurons and (2) the size of the input to each DAN grows, the speed, or efficiency, with which the Memory-Loss is minimized is increased. In other words, vectorized connections accelerated optimization of our meta-objective [Camp et al. 2020].



(a) (b)

Figure 5.5 a) Model definitions: **net0** uses a single, meta-learned phenotype, shared by all DANs, fixed during deployment; **net1** uses the same meta-learned single phenotype as **net0**, but it is fully plastic during deployment (updates to the φ are allowed); **net2** uses a random, shared phenotype, fixed during deployment; **net3** uses a random, shared phenotype, fully plastic during deployment; **net4** uses random, but completely unique DANs (no parameter sharing), fixed during deployment; **net5** uses random, but unique DANs, fully plastic during deployment. Once before learning begins, and after training on each successive task, the Total-Loss over the complete function is calculated. Clearly, the meta-learned phenotype outperforms random DANs. b)Total amount of Memory-Loss experienced by different models during deployment. Clearly, the meta-learned phenotype outperforms random DANs

5.7 Conclusions and Future Work

In this work, we offered a framework for thinking about artificial neurons as much more powerful functions, realized as deep artificial networks, which can be embedded inside larger plastic networks. We showed that it is possible to meta-learn a single parameter vector for such a model that, when held fixed, can facilitate a meta-objective during deployment. In the process, we hope to inspire a deeper understanding about the responsibilities of neurons in both artificial neural networks, as well as real brains. In future work, we plan to investigate the potential of DANs in real-world vision and reinforcement-learning settings, as well as the possibility of optimizing several meta-objectives at once.

CHAPTER 6

Hyper-Learning with Deep Artificial Neurons

6.1 Introduction

Neurons in real brains are enormously complex computational units. For a long time, the AI community has dismissed the prospect that this complexity can be leveraged in order to endow artificial networks with new capabilities that are not already achievable with traditional architectures that employ simple sum-and-fire neurons. Here, we offer evidence to the contrary, and demonstrate the benefits of dramatically increasing the power of individual neurons. We build upon prior work with Deep Artificial Neurons (DANs), which are themselves realized as deep neural networks. Networks of DANs are therefore composed of learned networks (computers) which are embedded within, and distributed throughout larger networks of randomly initialized plastic synapses. Like real neurons, DANs of the same type share common parameters that govern their behavior. In this work, we demonstrate that they can be explicitly trained to minimize some of the shortcomings of off-the-shelf optimizers and Backpropagation, such as catastrophic forgetting and poor OOD generalization in few-shot settings. DANs can therefore be said to regularize, or self-supervise, synaptic updates by warping the gradients flowing backwards through the network. We also introduce an end-to-end algorithmic framework for training and deploying networks of this type, called Hyper-Learning. In short, Hyper-Learning leverages population-based self-supervised meta-learning in order to teach DANs how to facilitate multiple meta-objectives during deployment. We offer empirical results which demonstrate near perfect memory retention

and competitive generalization on several non-iid, few-shot data-streams, under a variety of replay-free continual learning paradigms; including fully supervised incremental classification, fully unsupervised incremental learning, semi-supervised incremental learning, and incremental regression tasks.

6.2 Prior Work

It has been argued "that much of an animal's behavioral repertoire is not the result of clever learning algorithms...but arises instead from behavior programs already present at birth...(and that) these programs arise through evolution, are encoded in the genome, and emerge as a consequence of wiring up the brain" [Zador 2019]. However, this intuition, apt as it may be, might raise more questions than it answers. Which priors, for example, need to be encoded in the genome? How are those priors realized in the organism? Since most contemporary AI research is concerned with how to best go about finding good synaptic weights and architectures for specific tasks, it makes sense that this bias would be reflected in recent literature [Beaulieu et al. 2020; Finn et al. 2017; Flennerhag et al. 2020; Javed and White 2019]. On the other hand, very little work has been done to investigate the potential of learning powerful priors for neurons.

Real neurons are really complex [Beniaguev et al. 2020; Guergiuev et al. 2017; Hawkins and Blakeslee 2004; Izhikevich 2007; Jones and Kording 2020; Kandel et al. 2000; Palavalli et al. 2020]. They transform electro-chemical vectors into outbound action potentials and synaptic updates. The strengths of their connections are defined not by single scalar values,

but by high-dimensional vectors representing the state of chemical concentrations, electrical potentials, and proximity. They grow new appendages (dendrites), and some cells even physically migrate in the brain [Rahimi-Balaei et al. 2018]. Conversely, artificial neurons, with few exceptions, are extraordinarily simple models which reduce this enormous biological complexity to an elementary sum and fire operation. We ask, might there be some benefit to dramatically increasing the computational power of individual neurons in artificial neural networks? Is it possible that the cause of a lot of innate behavior has been hiding in plain sight; not in the connectome, but inside the neurons themselves?

There is at least some consensus that memories and knowledge are stored in the connections (synapses) between neurons [Izhikevich 2007; Kandel et al. 2000]. The malleability of these connections is known as synaptic plasticity, and the rules governing how the strengths are updated is known as learning. ANNs have excelled largely because Backpropagation is good at learning configurations of synaptic weights for immediate, well-defined tasks [Wang and Raj 2017]. However, the algorithm has historically failed in non-iid settings where the training distribution evolves over time [Kirkpatrick et al. 2016].

On the other hand, the plasticity dynamics of real neurons, even on short time frames, are much more complex than many in the machine learning community like to acknowledge. Even more biologically faithful attempts to explain the underlying update rules have failed to adequately model the dynamics required for long-term knowledge preservation [Payeur et al. 2020].

Let us further review some material in 4 areas crucial to our interpretation of the problem:

Complex Neurons and Synapses, Innate Structural Priors and Meta-Learning, Continual Learning and Memory, and Learned Optimizers.

There have been some very recent attempts to model more complex neurons and synapses. However, suffice it to say, it has still not been fully explored in the artificial intelligence literature. Some have sought to manually enforce dendritic segregation as a means to facilitate multi-plexing of feed-forward and feed-back signals in deep artificial networks [Guergiuev et al. 2017; Hawkins and Ahmad 2016]. Deep Complex Networks aimed to increase the complexity of the signals being propagated by introducing complex-valued connections and activations [Trabelsi et al. 2017]. In [Ba et al. 2016], the authors also investigated the potential benefits of 2 sets of synaptic weights; one for fast retention of knowledge, and another for memory over longer time horizons. Beniaguev et al. showed that a multi-layer ANN can mimic much of the functionality of real neurons [Beniaguev et al. 2020]. Aguera y Arcas et al. advocated for learning complex priors for neurons and synapses [Aguera y Arcas 2302], and a similar approach was advocated in [Gregor 2020] specifically for continual learning. Jones et al. demonstrated that single neurons may be computationally capable of solving sophisticated vision tasks [Jones and Kording 2020]. Mordvintsev et al. showed that it may be beneficial to think of individual cells as networks with self-organizing properties and high-dimensional message passing capabilities [Mordvintsev et al. 2020]. In [Randazzo et al. 2020], the authors expanded on this notion by formalizing a Message Passing Learning Protocol, whereby learning is facilitated through the communication of messages, realized as vectors, passed amongst nodes.

6.2.1 Innate Structural Priors and Meta-Learning

Many in the machine learning community have begun to at least acknowledge that efficient learning in real brains may result, not from clever learning algorithms, but from innate priors which have arisen through evolution, and which are generally kept fixed during intra-life deployment [Zador 2019]. Weight-Agnostic Neural Networks demonstrated that there exist architectures, not specific weight configurations, which exhibit out-of-the-box propensity for certain tasks [Gaier and Ha 2019]. The Lottery Ticket Hypothesis has shown that there often exist smaller, easily optimizable networks embedded in the weight matrices of much larger networks [Frankle and Carbin 2018], again hinting at the existence of optimal structure for specific tasks. As such, here have been numerous attempts to meta-learn a good prior over the weight distributions and architectures of ANNs. MAML aims to find parameters suitable for fast adaptation and few-shot learning [Finn et al. 2017]. In [Huang et al. 2020], the authors propose a technique to meta-learn a single, shared policy network which is distributed throughout the anatomy of a robot, allowing one network to control the behavior of multiple body-parts via message-passing.

6.2.2 Continual Learning: Memory Retention, Transfer, and Generalization

Generally speaking, machine learning optimization algorithms rely upon the *iid* assumption, which asserts that the training and test distributions need to be (approximately) the same. For this reason, the traditional, and most antiquated, technique for overcoming catastrophic forgetting is one which explicitly enforces this assumption, often referred to as experience

replay (ER). ER aims to ensure that the system is exposed to data drawn uniformly from the underlying task distribution/s. Continual Learning, however, requires memory retention even in settings where the current training distribution may not be representative of the total, historical data distribution; thus requiring a *non-iid* assumption. Thus, experience replay mitigates catastophic forgetting by *ignoring* the problem altogether, by converting a non-iid data-stream into a randomly shuffled dataset, which can be sampled from under the iid assumption. In the following sections, we argue that this is likely extremely misguided, and advocate for more research on replay-free continual learning.

Memory and Generalization are more strongly related than many would first presume. If one accepts that historical, previously encountered data is discarded, and not retained in a buffer, then the dual objectives of strong generalization and memory retention are actually quite similar. In this scenario, there exists a dataset D_H of all previously encountered data, as well as another dataset D_G, comprised of all unseen instances of previously seen classes (or tasks). Under the continual learning paradigm, as the data distribution evolves over time, historical data in D_H is likely to move further and further away from the current training distribution, to which the model is presently being exposed. Thus, the historical data is itself OOD, and the retention of memory requires generalization to this unavailable dataset D_H. Fortunately, prior work has shown that it is possible to explicitly optimize for memory retention in scenarios where previously encountered data is discarded. In fact, as we further demonstrate in the coming sections, it is remarkably easy to explicitly train a model to retain memory in replay-free settings. This is because the OOD dataset D_H is *known*,

and we can therefore teach a model how to recognize what knowledge it needs to preserve. This is exciting because it shows that, while its challenging, strong generalization can be incentivized and facilitated with the right training objective, and access to the right data distributions. In the coming sections, we present an algorithmic framework called Hyper-Learning that unifies and explicitly optimizes for the dual objectives of replay-free memory retention and strong few-shot generalization in non-iid settings.

Broadly speaking, 4 categories of techniques have emerged to combat catastrophic forgetting: (1) regularization-based approaches [Farajtabar et al. 2019; Kirkpatrick et al. 2016; Lesort et al. 2019], (2) knowledge-compression, or capacity expansion [Joseph and Balasubramanian 2020; Mandivarapu et al. 2020b; von Oswald et al. 2019b; Yoon et al. 2018], (3) experience replay and/or external memory slots [Graves et al. 2014; Mnih et al. 2013; Rolnick et al. 2019; Weston et al. 2015], and (4) meta-learned representations and update rules [Beaulieu et al. 2020; Flennerhag et al. 2020; Gregor 2020; Javed and White 2019; Lindsey and Litwin-Kumar 2020]. We build upon prior work that draws strong inspiration from a promising approach known as Warped Gradient Descent (WGD). WGD mitigates catastrophic forgetting by learning warp parameters ω, realized as warp-*layers*, which are interleaved between the standard layers of a neural network. These warp parameters are meta-trained and held *fixed* during deployment, allowing the rest of the network to learn, without forgetting, using standard Back-propagation. As the name implies, parameters ω warp activations in the forward pass, and the gradients in the backwards pass. The technique is promising because it offers guarantees of convergence, and a path towards continual-learning

at scale. In contrast to WGD, we show in our initial experiments that it is possible to learn a single, small network φ, constituting a single neuronal phenotype, that can be distributed throughout a larger plastic network, and which also mitigates catastrophic forgetting.

6.3 Hyper-learning Explained

Hyper-Learning is a population-based meta-learning framework for explicitly training DANs to facilitate the dual objectives of replay-free memory retention and strong OOD generalization on non-iid few-shot data-streams. Unlike the traditional machine learning paradigm, wherein parameters are learned and then held fixed during inference, we wish to learn parameters (of DANs) which can be fixed during *learning* in order to regulate the updates to another set of parameters (Synapses).

One of the key insights leveraged by Hyper-Learning is that even though we wish our models to be able to learn on non-iid data-streams, DANs must be trained under the iid assumption, since we still rely upon stochastic gradient-based optimization to learn their parameters. In other words, we seek DANs which, when held fixed, negate the necessity for the iid assumption when the model is deployed. To accomplish this, we train a population of M unique models on M non-iid data-streams. All models share common DANs, parameterized by φ, but each has unique synapses, defined by θ, which together define the complete state of each model at any given time-step. Note that since DANs are distributed throughout each model, the gradients for Synapses ∇_θ depend on DAN parameters φ, and the Hyper-gradients for DANs ∇_φ depend on the Synapses of ***all*** models θ_M. This is true,

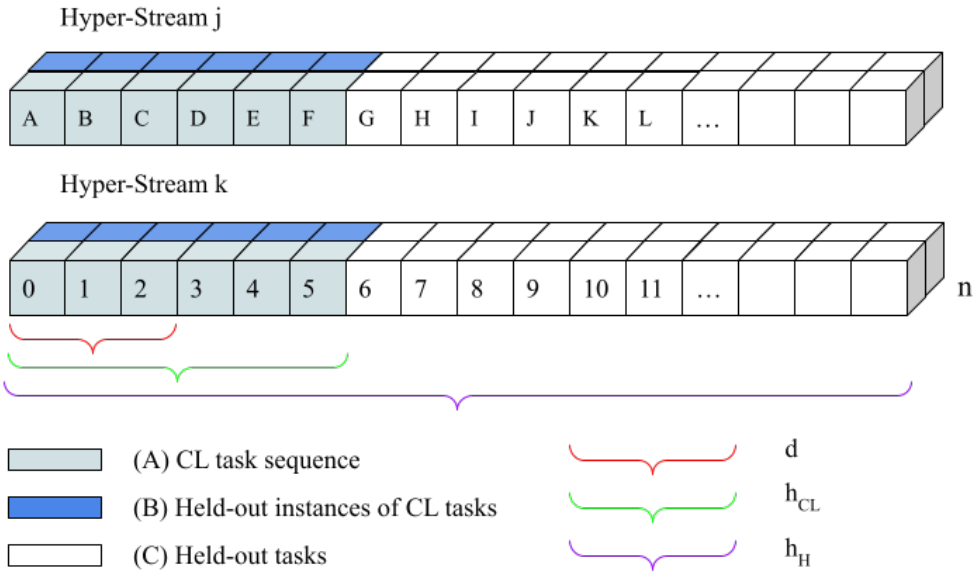


Figure 6.1 Two Hyper-Streams. Hyper Streams are comprised of sequences of tasks that must be learned in sequential fashion.

since each set of parameters θ and φ are factors of both sets of gradients. This distinction will come in handy w.r.t. notation when we quantify the losses used to update each set of parameters.

We begin by defining a large Hyper-Task distribution $p(\mathcal{T}_H)$, and several hyper-parameters (d, h_H, and h_{CL}) governing how $p(\mathcal{T}_H)$ is partitioned into M Hyper-Streams, as outlined in Algorithm 1. Under this framework, we assume the maximum number of tasks d that the model is expected to encounter when it is deployed, after Hyper-Training. This can be regarded as the deployment capacity of the model. Critically, the number of tasks on each Hyper-Stream is significantly *larger* than d, s.t. $d << h_{CL} << h_H$, where h_{CL} is the amount of tasks that the model must try to learn continually during each Hyper-Training epoch, and h_H is the total number of tasks on the Hyper-Stream. We refer to this as ***over-training***,

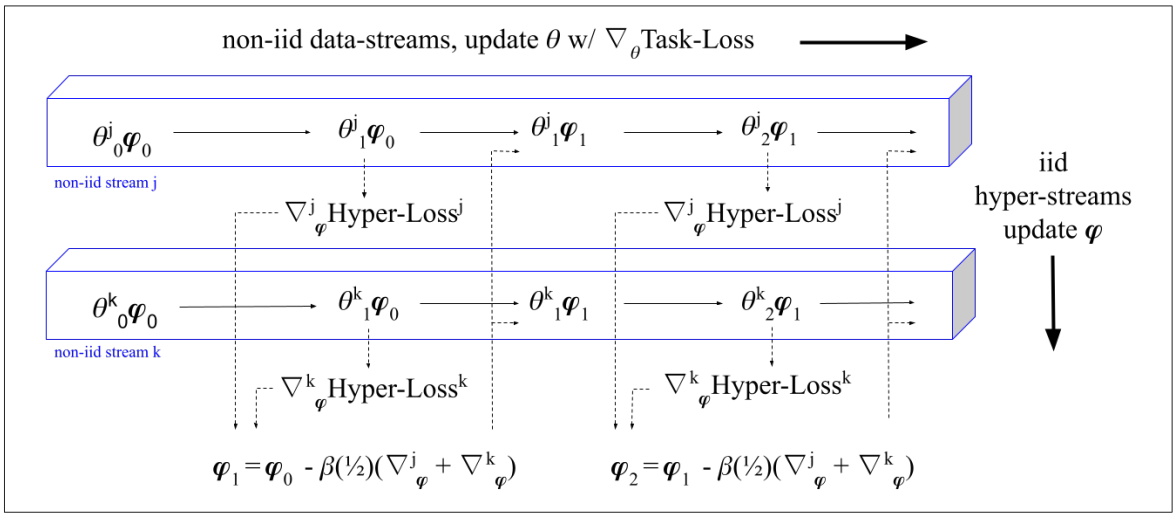


Figure 6.2 Hyper-Learning: A population of n=2 models is trained on 2 non-iid data-streams. Synapses are only ever exposed to the non-iid data on their respective streams, but DANs are shared across all n models, and are therefore trained under the iid assumption. This is critical since both the data and the Synapses on each stream evolve in a non-iid way. During Hyper-Training, gradients for DANs are computed and averaged across all models after each and every Synaptic update. Thus, DANs are explicitly trained to facilitate one or more meta-objectives during deployment. They are held fixed, while Synapses update using standard Back-propagation. In essence, Hyper-Training uses experience replay to teach a model how to learn efficiently and continually without it.

and it is crucial for achieving strong generalization during deployment. Each Hyper-Stream $\mathcal{H}_m$ (Figure 6.1) for each model $m \in M$, is therefore partitioned into a sequence of continual learning tasks, held out *instances* of tasks in the CL sequence, and a large batch of held-out tasks that Synapses are *never* exposed to, but which are used to quantify a generalization loss, as we explain in the coming sections.

At the outset of Hyper-Training, all models are randomly initialized to the same initial state, with Synapses θ_0 and DANs φ_0. Then, for each outer-loop epoch, we proceed by sampling a Hyper-Stream $(\mathcal{H}_m \subset H_M) \sim p(\mathcal{T}_H)$, of length $(d \times h_H)$, for each model m in the population M (Figure 6.1). As mentioned above, each Hyper-Stream contains a sequence

of Continual Learning Tasks which the associated model must try to learn in succession, without revisiting or forgetting tasks previously encountered on the stream. Here, a task is defined defined as a small few-shot batch of images from a single class i. The models are allowed to take k steps on each task in their continual learning sequence $\mathcal{H}_m^{CL} \subset \mathcal{H}_m$, and at each step t attempts to output the correct class label $\hat{y}$. Upon generating a class prediction $\hat{y}$, a Task-Loss is computed, and used to update the Synapses of each model independently. In the case of Supervised Learning, the Negative Log-Liklihood can be used, for example; or in the case of Unsupervised Learning, the mean squared error:

$$\mathcal{L}_m^{T_i} = NLL((\theta_t^m \varphi_t)(T_i)) \tag{6.1}$$

$$\mathcal{L}_m^{T_i} = MSE((\theta_t^m \varphi_t)(T_i), T_i) \tag{6.2}$$

Given the current Task-Loss $\mathcal{L}_m^i$, Synaptic gradients are computed, and each model is updated independently:

$$\theta_{t+1}^m \leftarrow \theta_t^m - \alpha \nabla \theta_t^m \mathcal{L}_m^i \qquad \forall\, m \in M \tag{6.3}$$

This results in new model states $\theta_{t+1}^m \varphi_t$. Note that this update may have caused forgetting over $\mathcal{H}_m^{CL}$, and, more generally, poor generalization over all of $\mathcal{H}_m$. Therefore, we wish to quantify and minimize a generalization loss, and a memory loss, simultaneously, by sampling a batch of tasks uniformly from the full Hyper-Stream: $\mathcal{H}_m^G \subset \mathcal{H}_m$. This Hyper-Loss can be

Hyper-Learning with Deep Artificial Neurons

Require: $p(\mathcal{T}_H)$: Hyper-Training task distribution
Require: M: Population of Models
Require: d: Assumed # deployment tasks
Require: h_H: Total # of tasks on each Hyper-Stream
Require: h_{CL}: # Continual Learning tasks during Hyper-Training, where $d << h_{CL} << h_H$
Require: k: steps_per_task
Require: α, γ: learning rate hyperparameters, for θ and φ respectively
for model m in M **do**
    $\theta \leftarrow \theta_0, \varphi \leftarrow \varphi_0$: randomly initialize all models to the same initial state
end for
while not done **do**
    for model m in M **do**
        Sample Hyper-Stream ($\mathcal{H}_m \subset H_M$) $\sim p(\mathcal{T}_H)$, of length ($d \times h_H$)
    end for
    for task_index i in range(h_{CL}) **do**
        for step t in range(k) **do**
            for model m in M **do**
                Cur_Task $\leftarrow (T_i \in \mathcal{H}_m)$
                Compute $TaskLoss$ $\mathcal{L}_m^i$ w.r.t task Cur_Task
                Compute Synapse gradients $\nabla\theta_t^m$ via Backprop, w.r.t. $\mathcal{L}_m^i$
                Update Synapses: $\theta_{t+1}^m \leftarrow \theta_t^m - \alpha\nabla\theta_t^m$
                Sample Random Task Batch from full Hyper-Stream $\mathcal{H}_m^G \subset \mathcal{H}_m$
                Given θ_{t+1}^m, Compute $HyperLoss$ $\mathcal{L}_m^G$ w.r.t $\mathcal{H}_m^G$
                Compute DAN gradients $\nabla\varphi_t^m$ via Backprop, w.r.t. $\mathcal{L}_m^G$
            end for
            Average DAN gradients across all models
            Update DANs in all models: $\varphi_{t+1}^M \leftarrow \varphi_t^M - \gamma\nabla\varphi_t^M$
        end for
    end for
    $\theta \leftarrow \theta_0$: reset Synapses to initialization
end while

computed by measuring the associated model's predictions over this batch of tasks, which likely includes some tasks already encountered in the Continual Learning sequence:

$$\mathcal{L}_m^G = NLL((\theta_{t+1}^m \varphi_t(\mathcal{H}_m^G))) \qquad \forall\, m \in M \tag{6.4}$$

Or, for Unsupervised Class Incremental Learning:

$$\mathcal{L}_m^G = MSE((\theta_{t+1}^m \varphi_t(\mathcal{H}_m^G)), \mathcal{H}_m^G) \qquad \forall\, m \in M \tag{6.5}$$

This loss can be minimized by computing and averaging the gradients for DANs, across all models:

$$\varphi_{t+1} \leftarrow \varphi_t - \gamma(\frac{1}{m}) \sum^{M} \nabla \varphi_t \mathcal{L}_m^G \qquad \forall\, m \in M \tag{6.6}$$

At the end of each Outer-Loop epoch, the synapses of each model are *reset*, and new Hyper-Streams are sampled, before repeating the process described above.

After Hyper-Training is completed, the model is *deployed*. DAN parameters φ are held ***fixed***, and the model is obligated to learn efficiently and continually, without forgetting, using standard backpropagation. We offer empirical validation of our approach in the next section.

6.4 Hyper-Learning Results

Formally, for both the Supervised and Unsupervised Class Incremental settings, we define a task sequence as i instances from c ordered classes. When working with static image datasets, We can convert this task (or class) sequence to something more analogous to a video stream by duplicating the i instances for k frames. Consider, for example, an image of the digit 1, as might be sampled from the MNIST dataset. We can construct a video-stream by duplicating this image k times, resulting in a stream of length k. If we repeat this step for all c classes, we can therefore create video-streams of length $(c \times k)$. In practice, we assume

all instances of a class i are consumed as a batch, as if the model is being exposed to i separate instances of the full hyper-stream, where the underlying order of the task-sequence remains unchanged. Additionally, recall that d is the assumed number of deployment tasks, h_{CL} is the number of Continual Learning tasks in a sequence, and h_H is the *total* number of tasks on *each* Hyper-Stream. For this experiment. We set $k = 25$, $i = 5$, $d = 10$, $h_H = 25$, and $h_{CL} = 20$. Thus, $d << h_{CL} << h_H$. That is, each Continual-Learning task sequence on a Hyper-stream is of length $h_{CL} \times k = 20 \times 25 = 625$. In other words, during Hyper-Training, the models are explicitly optimized to retain memory and converge to strongly generalizing parameter configurations over 625 synaptic updates.

6.4.1 Supervised Class Incremental Learning

For the experiments depicted in Figure 6.3, the model is Hyper-Trained on a subset of Omniglot, and then deployed for evaluation on (a) a held-out subset of Omniglot (Meta-Test), and (b) MNIST (Meta-Eval), During deployment, the model is obligated to learn a sequence of tasks in an online, or replay-free manner. That is, the data from each task is encountered once, and only once. The Omnigot dataset is divided into two subsets. One is used as the Hyper-Training distribution, and the other is used for Meta-Testing. A population of 3 models is Hyper-Trained the distribution which is constructed according to the Hyper-Learning Algorithm defined in Section 6.3.

For this experiment, we used a model size of 10 million parameters, and learning rates of .001 for both sets of parameters θ and φ. As Figure 6.3 shows, both memory retention and generalization *after* continual learning on a sequence of 10 tasks steadily improves as Hyper-

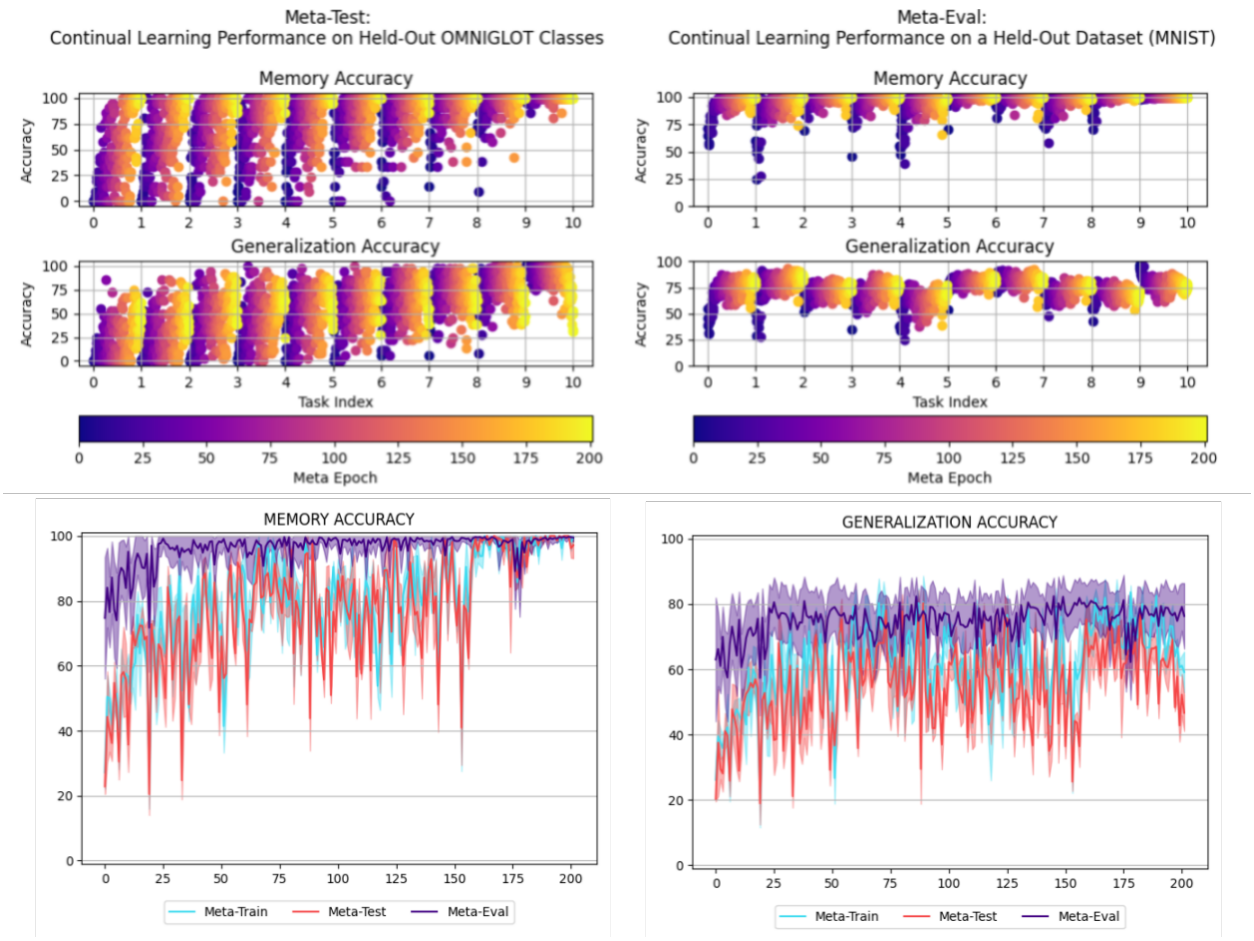


Figure 6.3 Results: Supervised Class Incremental Learning. Meta-Test = Held-Out Omniglot; Meta-Eval = MNIST. (Top) As Hyper-Training proceeds, the models's memory w.r.t to each task in the CL sequence is improved (pulled up), and is maintained for the entire task-learning sequence. (Bottom) The average memory and generalization accuracies are averaged computed and averaged over all tasks *after* the model has learned all 10 tasks.

Training progresses, though generalization improves at a much slower rate, and does not appear to converge. These results served as promising baselines upon which to improve the models and their training protocols. As the coming sections will show, we hypothesized that some of the usual techniques would allow us to smoothly improve the overall performance: more data, and bigger models.

6.4.2 Unsupervised Class Incremental Learning

We then used the same pre-processing and algorithmic framework to evaluate the ability of our models to perform *Unsupervised* Class Incremental Learning on the same task distributions as those defined in the previous section on Supervised Learning. As Figure 6.4, Hyper-Training effectively endows networks of this type to be able to perform unsupervised learning on a sequence of tasks in an online manner. For this experiment. We used the MSE variant of the Hyper-Loss defined in Equation 6.5. Additionally, we doubled the size of the model to 20 million parameters by introducing a Decoder, resulting in a more traditional Auto-Encoder architecture, or more aptly, a DAN-Auto-Encoder. We posit that the increased model size might account for how well this model is able to generalize. As Figure 6.4 clearly demonstrates, the Hyper-Trained model exhibits excellent memory retention *and* forward transfer during online continual learning, as evidenced by the fact that reconstructions of unseen downstream classes improves dramatically after only performing learning over the first 5 digits.

More clearly, after Hyper-Training, the model was deployed on MNIST, and was obligated to learn to reconstruct $s = 5$ instances of all 10 digits, seen consecutively. That is, during deployment, the model is allowed to learn 0's, before moving on to 1's, and then 2's, and so on. The model was allowed to perform 60 steps of gradient descent on the s samples from each class, where each class is seen only once, and the model is **not** permitted to revisit any old data. The middle image in Figure 6.4 shows that the model improves at reconstructing unseen digits as it learns to reconstruct each current digit, i.e. it exhibits forward transfer.

Figure 6.4 Results: Unsupervised Class Incrental Learning. Replay-free memory retention and transfer during deployment on MNIST (after meta-training on Omniglot). (Left) The model's initial attempt to reconstruct it's input (MNIST digits), after a single step of gradient descent on 0's. (Middle) The model has performed 60 steps of gradient descent on each digit 0-4, seen consecutively, 4's most recently. (RIGHT) Reconstructions of all previously seen digits (0-9) after training on all digits, 9's most recently.

Note that our algorithm explicitly encourages transfer by quantifying it during computation of the Hyper-Loss. By measuring the model's performance over the full task sequence after each and every inner-loop step, the model is trained to learn information at every step that may improve its performance on future downstream tasks.

6.4.3 Improving Generalization with Over-Training

In this section, we offer initial results which demonstrate the benefits of ***over-training***. We proceed by first fixing our deployment task-sequence lengths $d = 3$ tasks, which must be learned in succession. For the experiments depicted in Figures 6.5, 6.6, and 6.7 we use a model size of 10 million parameters, learning rates of .001 for θ and φ, We then proceeded to investigate the effect of increasing the amount of data seen during Hyper-Training on downstream performance, as defined by memory retention and generalization *after* continual

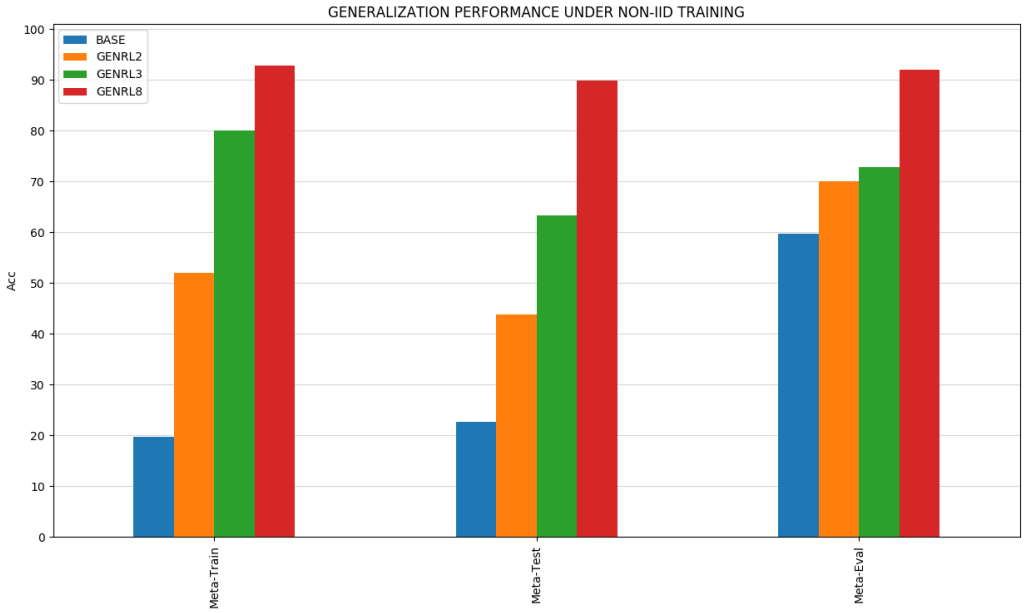


Figure 6.5 The benefits of Over-Training during Hyper-Training

learning. As Figure 6.5 clearly shows, performance smoothly improves as the amount of data is increased. That is, in order to increase the amount of data used during Hyper-Training, we can increase the total number of tasks on the Hyper-Stream h_H, the length of the continual learning task sequences on each Hyper-Stream h_{CL}, or the number of class instances i on each Hyper-Stream. We began by establishing a baseline performance; where $i = 1$, $h_{CL} = 3$, and $h_H = 5$ (see Blue Bars in Figure 6.5). We then proceeded to scale the amount of data used during Hyper-Training in the following way:

- **Orange**: $i = 5$, $h_{CL} = 6$, and $h_H = 10$
- **Green**: $i = 10$, $h_{CL} = 10$, and $h_H = 15$
- **Red**: $i = 20$, $h_{CL} = 20$, and $h_H = 25$

Figure 6.6 shows the memory retention of the final over-trained model during Hyper-Training. Near-perfect memory retention is quickly attained by over-training the models.

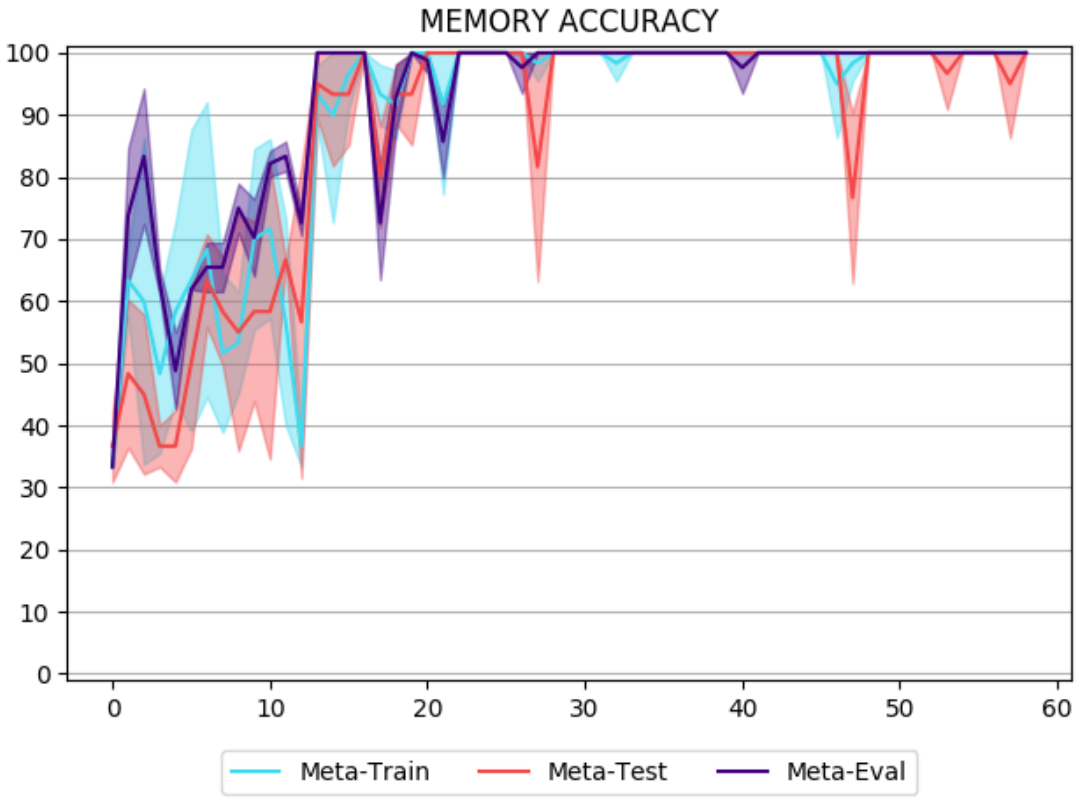


Figure 6.6 Over-Training Memory Accuracy during Hyper-Training

As the amount of instances for each class, the length of the continual-learning sequences, and the total number of classes on a Hyper-Stream increase, the generalization performance during deployment, and after continual-learning smoothly improves. Thus, when model size is held as a constant, there is a clear benefit to increasing the amount of data used during Hyper-Training. Further, this trend clearly hints at the more general relationship between

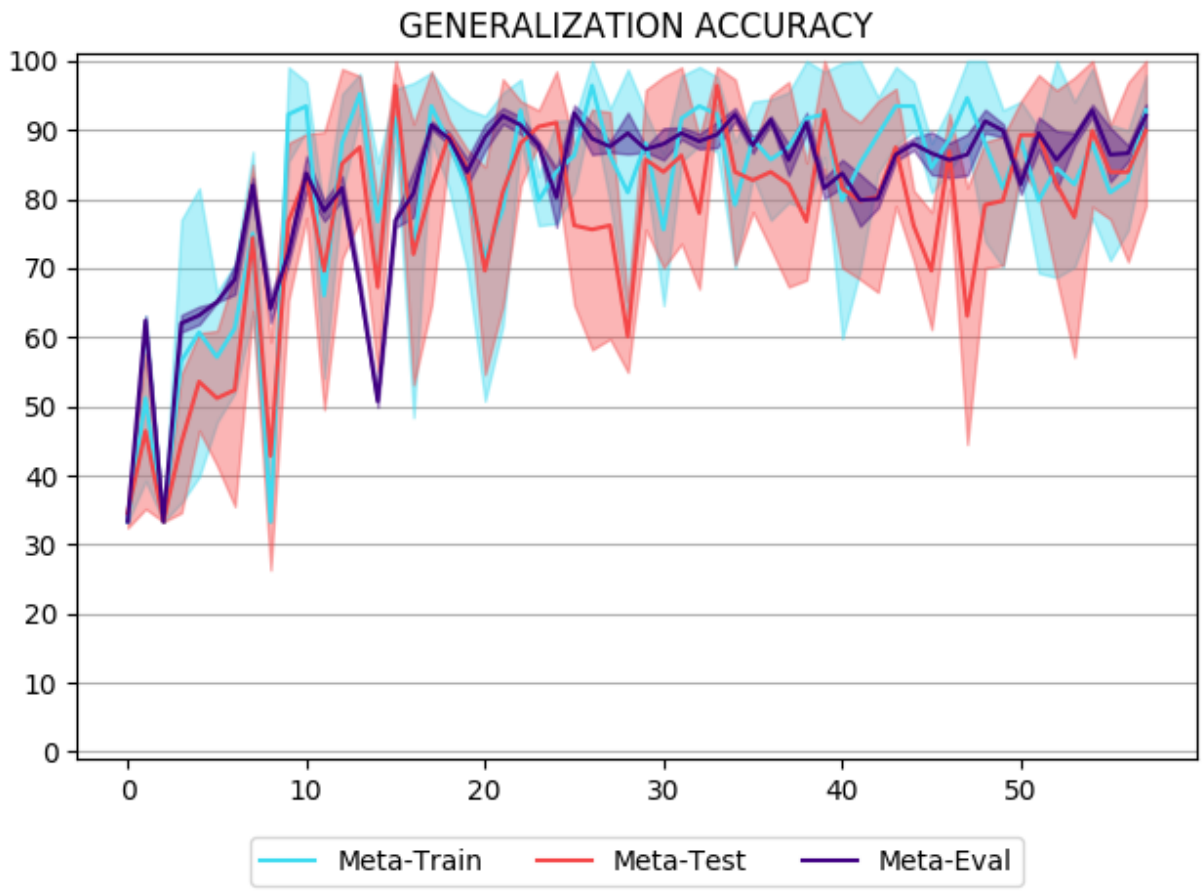


Figure 6.7 Over-Training Generalization during Hyper-Training

model size (aka capacity), the amount of data available during Hyper-Training, and the downstream continual-learning abilities of the model.

Figure 6.7 shows the generalization perormance of the final over-trained model during Hyper-Training. Note that generalization accuracy is near 90% *after* continual-learning over the 3 deployment tasks, for all 3 datasets (Meta-Train, Meta-Test, and Meta-Eval), which are defined in the preceding sections.

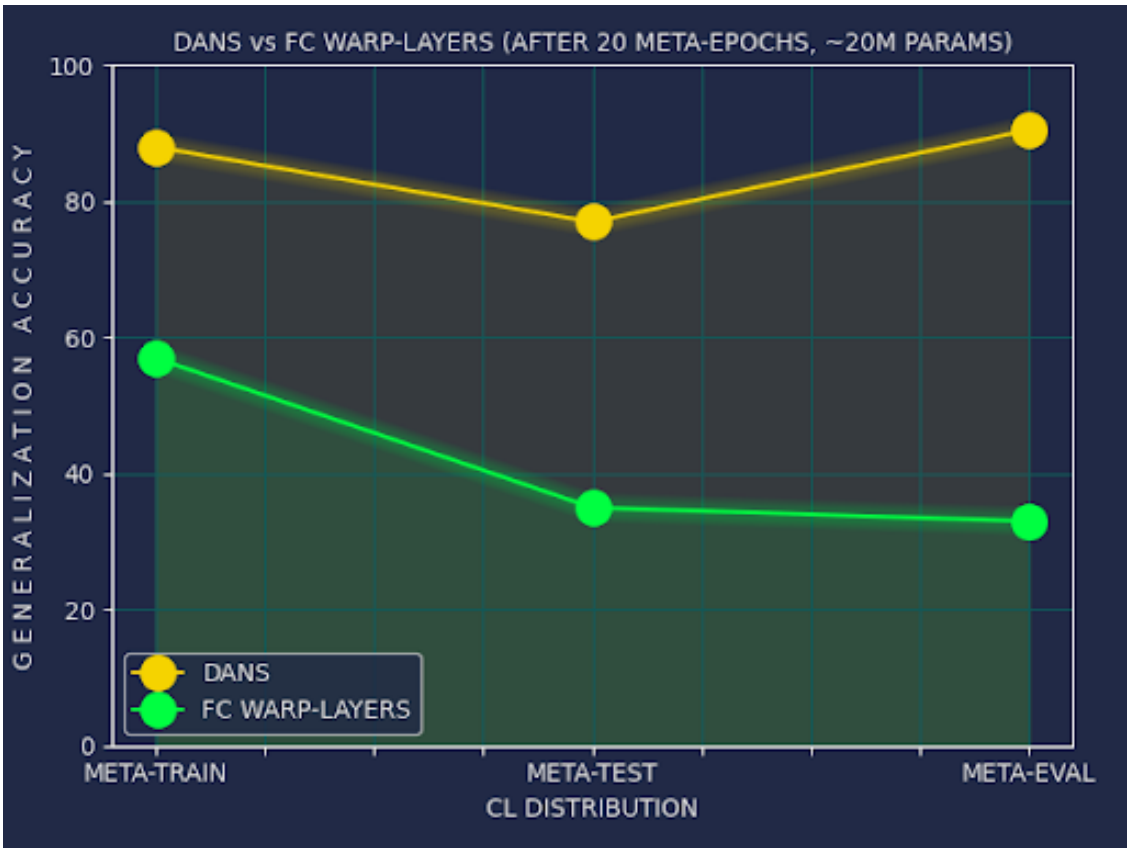


Figure 6.8 The modularity and parameter sharing enforced by DANs dramatically outperform Fully-Connected Linear Warp-Layers.

6.4.4 DANs vs Fully-Connected Warp-Layers

We next sought to establish whether the modularity and parameter-sharing enforced by a Network of DANs actually improves performance when compared to the fully-connected warp-layers used by Flennerhag et al. [2020]. Figure 6.8 clearly shows that DANs *dramatically* fully-connected layers. To make sure the comparison was a fair one, we took care to ensure that the total number of parameters in both models were roughly the same. Recall that all DANs within a payer share parameters. Also, note that DANs themselves contain several layers of non-linearities. Therefore, although fully-connected weight matrices may be

significantly *wider* than DANs, DANs are several layers *deeper* than their fully-connected counterparts. Consequently, it is relatively straightforward to enforce similar model sizes across both variants.

The results shown in Figure 6.8 demonstrate that Hyper-Training with DANs is far more efficient than using fully-connected warp layers. Clearly, the additional innate non-linearities are endowing the networks with a strong inductive bias that can be leveraged to learn more efficiently.

6.4.5 Improving Performance with Model Scale

In recent years, the artificial intelligence community has witnessed an explosive growth in the capabilities of deep learning systems. This profound jump in performance can be attributed, in large part, to a single variable: model size [Kaplan et al. 2020]. Consequently, we hypothesized that we would see a predictable improvement in performance as we increased the scale of our models. Figure 6.9 provides validation of that hypothesis.

We trained 3 populations, each comprised of 5 models, on the same underlying Hyper-Training and Deployment distributions. The smallest models had 25 million parameters, the next population contained models with 75 million parameters, and the largest models contained approximately 200 million parameters. Note that a population of 5 models with 200 million parameters each results in over 1 Billion trainable parameters in the population.

For these experiments, the Hyper-Training distribution was composed of a mixture of the following Datasets: Omniglot, FashionMNIST, CIFAR100, and mini-ImageNET. Hyper-Streams were constructed by sampling sequences of tasks from these underlying datasets

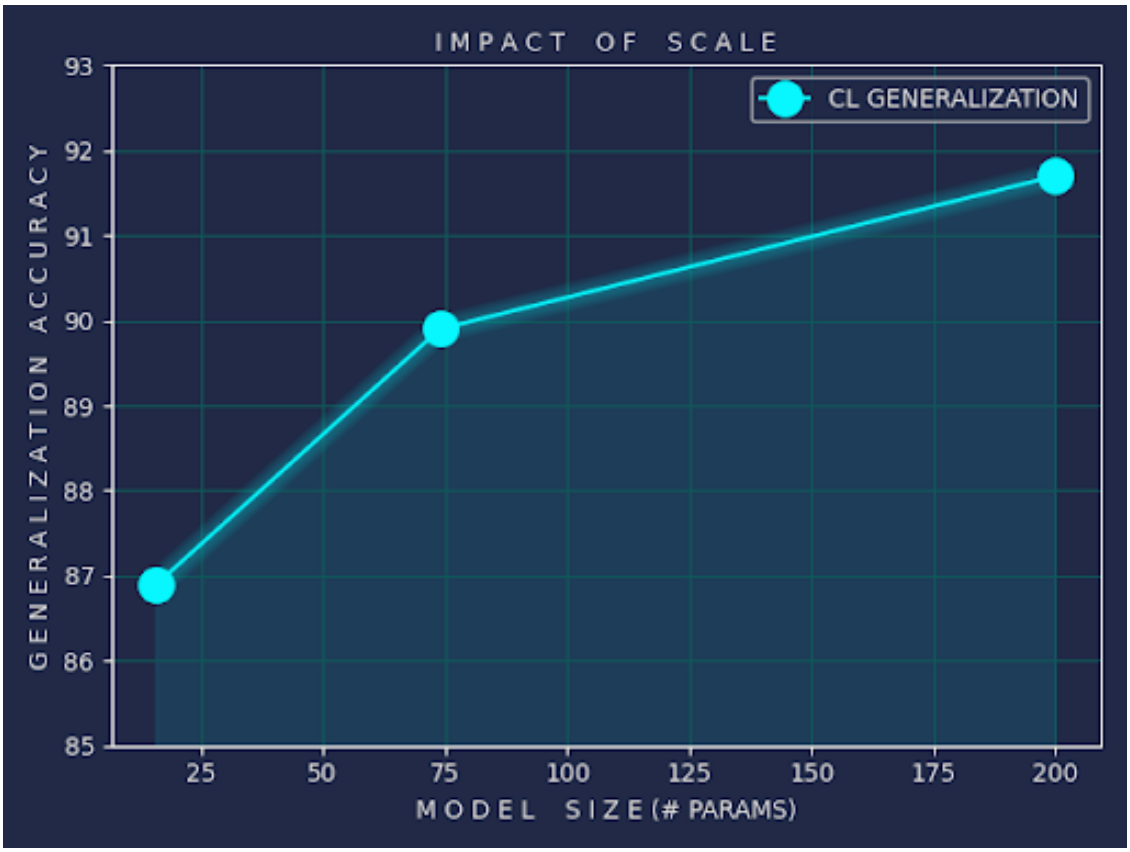


Figure 6.9 Scaling Trends

according to the protocol outlined in section 6.3. We then evaluated the performance of an individual model after being deployed on several continual-learning sequences drawn from (a) Meta-Train: the Hyper-Training distribution itself, (b) Meta-Eval: a heldout subset of Omniglot, and (c) MNIST. Figure 6.9 shows the generalization performance of models of various sizes after being obligated to learn all 10 MNIST digits in sequential fashion. The largest models, containing 200 million parameters achieve a generalization of nearly 92% after performing supervised learning on a video-stream of MNIST digits. The video-stream itself contains 5 instances of each digit, which means that this is both a few-shot *and* a

continual-learning problem.

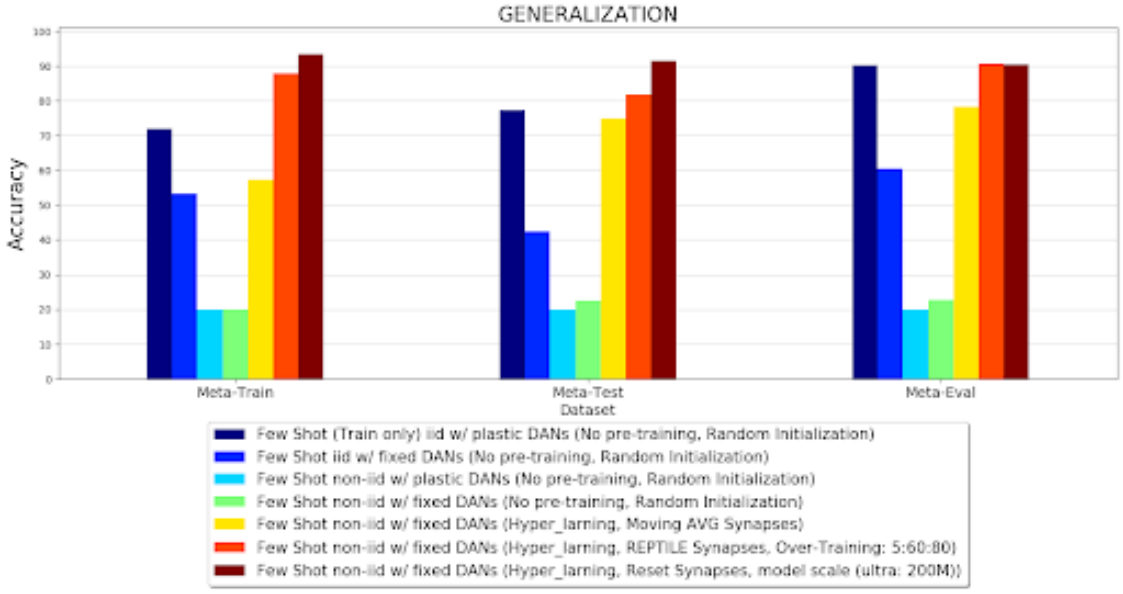


Figure 6.10 Generalization *after* continual learning. Hyper-Training results in models that are able to learn continually and in the process out-perform their randomly initialized counterparts that use iid-replay.

For many problems, the iid-assumption and replay-based approaches can be leveraged by artificial neural networks to obviate the necessity to learn in online manner. As such, the most reliable technique for overcoming catastrophic forgetting has been to retain copies of all data encountered by the system. Such a system is then able to sample uniformly from the underlying historical data distribution. It therefore serves as reasonable baseline against which to compare our models. The dark blue bar-plots in Figure 6.11 show the performance of iid-sampling under a fixed-compute budget. For this experiment, we tested the generalization of each model after being obligated to learn a sequence of tasks using a fixed budget of 500 gradient updates, occurring over 500 video frames. That is, the models

are only allowed 500 updates, 1 per frame of video. We tested several variants, as the legend clearly describes. The raw baseline (dark blue) shows the performance of a Network of DANs with 200M parameters that is randomly initialized and allowed 500 iid gradient updates.

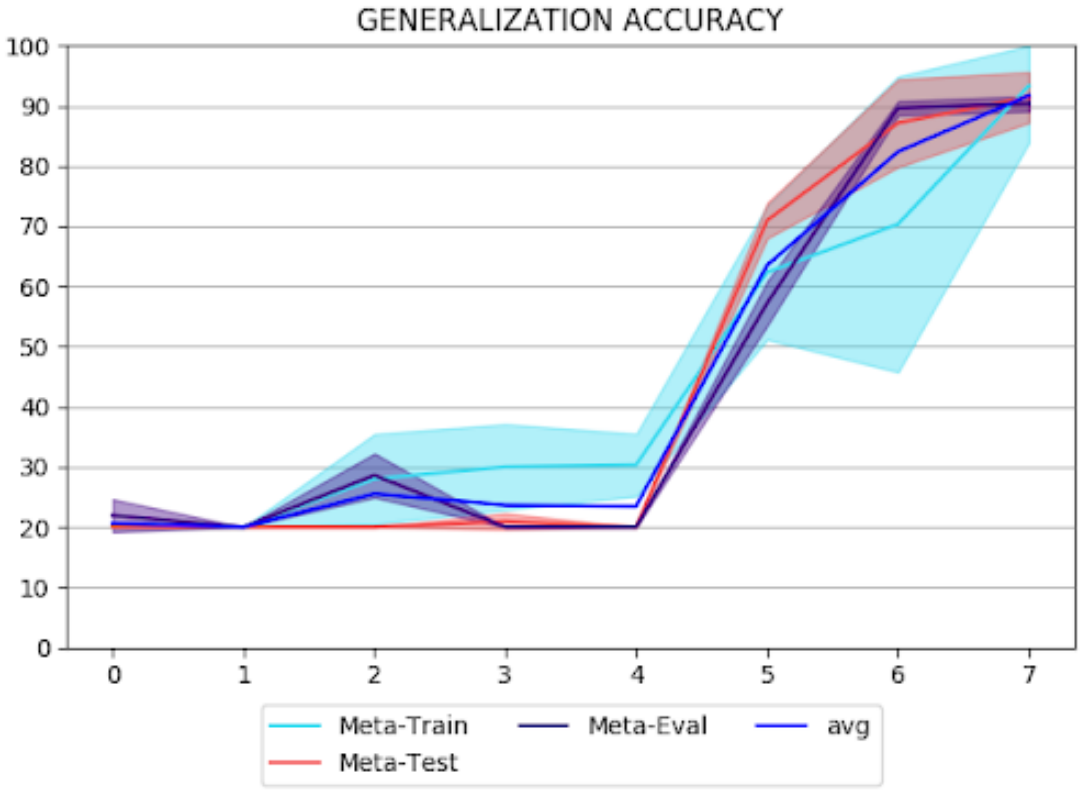


Figure 6.11 At scale, there are continual learning sequences within the hyper-training distribution for which: Memory Retention: 100% Generalization after CL: 100%.

Our results show that Hyper-Training, which is akin to clever pre-training, allows a model to significantly outperform the iid baseline with respect to generalization *despite* being obligated to learn the tasks in sequential (non-iid) fashion. Said another way, we can train models to outperform their randomly initialized iid baselines, but it comes at the ***cost*** of a lengthy Hyper-Training phase. As such, it is possible that this approach might be best

suited for situations in which the systems are simply not allowed to retain copies of the full evolving training distribution.

Furthermore, it is worth noting that there are indeed task-sequences within the Hyper-Training distribution upon which the models attain 100% generalization performance, as seen in Figure 6.11. This makes sense, since it ought to be possible to overfit to the Hyper-Training distribution itself. It should therefore be possible to further reduce the variance in generalization performance over the Hyper-Training distribution, but we leave this to future work.

6.4.6 DANs Are Learned Activation Functions

The astute reader may have realized that the combination of distributed, common parameters and Hyper-Training allows for an interpretation of DANs as ***learned action functions***. Our work shows that these learned activation functions play a critical role with respect to both memory retention and few-shot generalization. In order to more closely inspect them, we ran a continual regression experiment wherein the DANs took a single value as input and produced a single value as output. This allowed us to visualize the shape of the learned function itself. As Figure 6.12 shows, the learned activation function implemented by DANs can take on interesting shapes that are wildly dissimilar from most other common nonlinear activation functions used in deep networks (such ReLU, Leaky-RelU, Tanh, Sigmoid, etc...), despite being constructed using those very functions. Futher, the gradient of these learned functions also takes on interesting shapes.

While we leave a more thorough investigation to future work, we posit that these learned

function embed a strong inductive bias into deep networks that can facilitate multiple desirable meta-objectives when Hyper-Trained with a suitable data distribution.

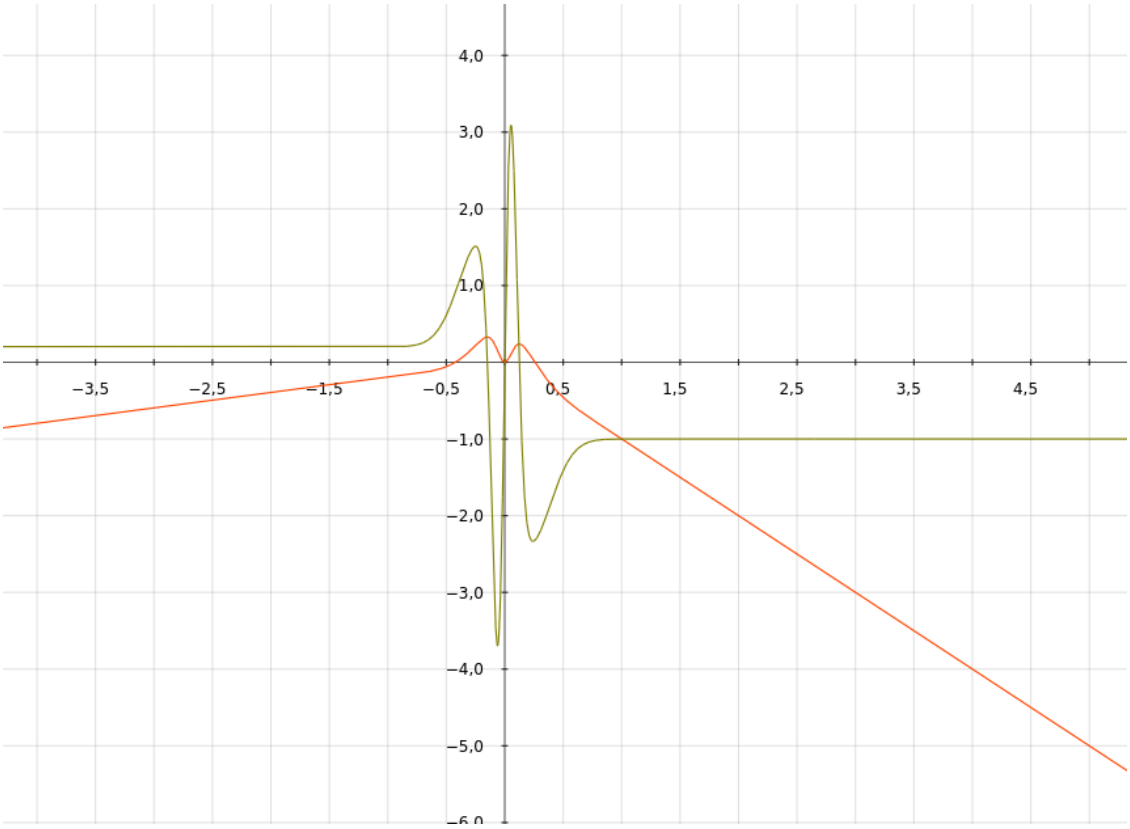


Figure 6.12 DANs are learned activation functions which have interesting derivatives. The feed-forward function is displayed in red, and its derivative is in brown.

6.5 Concluding Remarks and Future Prospects

Hyper-Learning with Deep Artificial Neurons represents the zenith of all my research done during my graduate program. It is this author's humble opinion that it is a framework that can be leveraged to tackle some of the most difficult open challenges in artificial intelligence. The research laid forth in this text should serve as a demonstration that if we can properly

define our problems of interest, we can either carefully design their solutions, whether they be architectural, or algorithmic. The appeal of Hyper-Learning with DANs, is that it is entirely compatible with all of the previous work outlined in this work, since DANs can be embedded in any arbitrary computation graph that we normally refer to as an artificial neural network. As such, several obvious options exist for promising future research. They include, but are not limited to:

- Hyper-Training DANs to facilitate Continual Active Learning by converting the proposed Barlow-Twins Active Learning Architecture to a Network of DANs.
- Converting open-source Large Language Models to Networks of DANs to allow of replay-free fine-tuning.
- Continual Alignment: Endowing Large Language Models with an ability to retain their alignment objectives through fine-tuning without open-sourcing the alignment data-set itself.

REFERENCES

Aguera y Arcas, B. (NeurIPS, 2019. URL https://slideslive.com/38922302.). Social intelligence.

Ba, J., Hinton, G., Mnih, V., Leibo, J. Z., and Ionescu, C. (2016). Using fast weights to attend to the recent past.

Beaulieu, S., Frati, L., Miconi, T., Lehman, J., Stanley, K. O., Clune, J., and Cheney, N. (2020). Learning to continually learn.

Beluch, W. H., Genewein, T., Nurnberger, A., and Kohler, J. M. (2018). The power of ensembles for active learning in image classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9368–9377.

Beluch, W. H., Genewein, T., Nürnberger, A., and Köhler, J. M. (2018). The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9368–9377.

Bengar, J. Z., van de Weijer, J., Twardowski, B., and Raducanu, B. (2021). Reducing label effort: Self-supervised meets active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 1631–1639.

Beniaguev, D., Segev, I., and London, M. (2020). Single cortical neurons as deep artificial neural networks. *bioRxiv*.

Camp, B., Mandivarapu, J. K., and Estrada, R. (2020). Continual learning with deep artificial neurons. *CoRR*, abs/2011.07035.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Dasgupta, S. (2011). Two faces of active learning. *Theoretical computer science*, 412(19):1767–1781.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv e-prints*, page arXiv:1606.05908.

Farajtabar, M., Azizan, N., Mott, A., and Li, A. (2019). Orthogonal gradient descent for continual learning.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400.

Flennerhag, S., Rusu, A. A., Pascanu, R., Yin, H., and Hadsell, R. (2020). Meta-learning with warped gradient descent. *ArXiv*, abs/1909.00025.

Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635.

Freeman, L. C. (1965). *Elementary applied statistics: for students in behavioral science*. John Wiley & Sons.

Gaier, A. and Ha, D. (2019). Weight agnostic neural networks. *CoRR*, abs/1906.04358.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing

model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org.

Geifman, Y. and El-Yaniv, R. (2017). Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*.

Gissin, D. and Shalev-Shwartz, S. (2019). Discriminative active learning. *arXiv preprint arXiv:1907.06347*.

Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306.

Gorriz, M., Carlier, A., Faure, E., and Giró-i-Nieto, X. (2017). Cost-effective active learning for melanoma segmentation. *CoRR*, abs/1711.09168.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines.

Gregor, K. (2020). Finding online neural update rules by learning to remember.

Greydanus, S. (2017). baby-a3c. `https://github.com/greydanus/baby-a3c`.

Guergiuev, J., Lillicrap, T. P., and Richards, B. A. (2017). Towards deep learning with segregated dendrites.

Guerguiev, J., Kording, K. P., and Richards, B. A. (2019). Spike-based causal inference for weight alignment.

Ha, D., Dai, A. M., and Le, Q. V. (2016). Hypernetworks. *CoRR*, abs/1609.09106.

Hanneke, S. et al. (2014). Theory of disagreement-based active learning. *Foundations and Trends® in Machine Learning*, 7(2-3):131–309.

Hawkins, J. and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10.

Hawkins, J. and Blakeslee, S. (2004). *On Intelligence*. Times Books, USA.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6.

Huang, W., Mordatch, I., and Pathak, D. (2020). One policy to control them all: Shared modular policies for agent-agnostic control.

Huszár, F. (2018). Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, 115(11):E2496–E2497.

Izhikevich, E. (2007). Dynamical systems in neuroscience. *MIT Press*, page 111.

Javed, K. and White, M. (2019). Meta-learning representations for continual learning. *CoRR*, abs/1905.12588.

Jones, I. S. and Kording, K. P. (2020). Can single neurons solve mnist? the computational power of biological dendritic trees.

Joseph, K. J. and Balasubramanian, V. N. (2020). Meta-consolidation for continual learning.

Kandel, E., Kandel, E., Schwartz, J., Jessell, J., Jessell, T., of Biochemistry, P., and Molec-

ular Biophysics Thomas M Jessell, M. (2000). Principles of neural science, fourth edition.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2016). Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796.

Kirsch, A., Van Amersfoort, J., and Gal, Y. (2019). Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32.

Kirsch, A., van Amersfoort, J., and Gal, Y. (2019). BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning. *arXiv e-prints*, page arXiv:1906.08158.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied

to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

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

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lesort, T., Stoian, A., and Filliat, D. (2019). Regularization shortcomings for continual learning.

Lewis, D. D. and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. In *Machine learning proceedings 1994*, pages 148–156. Elsevier.

Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. corr, abs. *arXiv preprint cmp-lg/9407020*, 16.

Li, X. and Guo, Y. (2013). Adaptive active learning for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 859–866.

Li, Z. and Hoiem, D. (2016). Learning without forgetting. *CoRR*, abs/1606.09282.

Lindsey, J. and Litwin-Kumar, A. (2020). Learning to learn with feedback and local plasticity.

Luo, W., Schwing, A., and Urtasun, R. (2013). Latent structured active learning. *Advances in Neural Information Processing Systems*, 26.

Mahapatra, D., Poellinger, A., Shao, L., and Reyes, M. (2021). Interpretability-driven sample selection using self supervised learning for disease classification and segmentation. *IEEE Transactions on Medical Imaging*, 40(10):2548–2562.

Mandivarapu, J. K., Camp, B., and Estrada, R. (2020a). Deep active learning via open set

recognition. *arXiv preprint arXiv:2007.02196*.

Mandivarapu, J. K., Camp, B., and Estrada, R. (2020b). Self-net: Lifelong learning via continual self-modeling. *Frontiers in Artificial Intelligence*, 3:19.

McCallum, A. and Nigam, K. (1998). Employing em and pool-based active learning for text classification. In *International Conference on Machine Learning*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. (2020). Growing neural cellular automata. *Distill*. https://distill.pub/2020/growing-ca.

Mundt, M., Majumder, S., Pliushch, I., and Ramesh, V. (2019a). Unified probabilistic deep continual learning through generative replay and open set recognition. *CoRR*, abs/1905.12019.

Mundt, M., Pliushch, I., Majumder, S., and Ramesh, V. (2019b). Open set recognition through deep neural network uncertainty: Does out-of-distribution detection require generative classifiers? *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 753–757.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2018). Variational continual learning. In *International Conference on Learning Representations*.

Palavalli, A., Tizón-Escamilla, N., Rupprecht, J.-F., and Lecuit, T. (2020). Deterministic and stochastic rules of branching govern dendritic morphogenesis of sensory neurons. *bioRxiv*.

Payeur, A., Guerguiev, J., Zenke, F., Richards, B. A., and Naud, R. (2020). Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *bioRxiv*.

Rahimi-Balaei, M., Bergen, H., Kong, J., and Marzban, H. (2018). Neuronal migration during development of the cerebellum. *Frontiers in Cellular Neuroscience*, 12:484.

Randazzo, E., Niklasson, E., and Mordvintsev, A. (2020). Mplp: Learning a message passing learning protocol.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. (2019). Experience replay for continual learning.

Roth, D. and Small, K. (2006). Margin-based active learning for structured output spaces. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, pages 413–424. Springer.

Scheffer, T., Decomain, C., and Wrobel, S. (2001). Active hidden markov models for information extraction. In *International Symposium on Intelligent Data Analysis*, pages 309–318. Springer.

Schohn, G. and Cohn, D. (2000). Less is more: Active learning with support vector machines. In *ICML*, volume 2, page 6. Citeseer.

Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Whye Teh, Y., Pascanu,

R., and Hadsell, R. (2018). Progress & Compress: A scalable framework for continual learning. *ArXiv e-prints*.

Sener, O. and Savarese, S. (2017). Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*.

Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, 52.

Settles, B. (2012). *Active Learning*, volume 6.

Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *proceedings of the 2008 conference on empirical methods in natural language processing*, pages 1070–1079.

Shui, C., Zhou, F., Gagné, C., and Wang, B. (2020). Deep active learning: Unified and principled method for query and training. In *International Conference on Artificial Intelligence and Statistics*, pages 1308–1318. PMLR.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

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

Tong, S. and Koller, D. (2001). Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66.

Trabelsi, C., Bilaniuk, O., Serdyuk, D., Subramanian, S., Santos, J. F., Mehri, S., Rostamzadeh, N., Bengio, Y., and Pal, C. J. (2017). Deep complex networks. *CoRR*, abs/1705.09792.

Tur, G., Hakkani-Tür, D., and Schapire, R. E. (2005). Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171–186.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. (2019a). Continual learning with hypernetworks. *CoRR*, abs/1906.00695.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. (2019b). Continual learning with hypernetworks. *CoRR*, abs/1906.00695.

Wang, G., Hwang, J.-N., Rose, C., and Wallace, F. (2017). Uncertainty sampling based active learning with diversity constraint by sparse selection. In *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE.

Wang, H. and Raj, B. (2017). On the origin of deep learning.

Wang, K., Zhang, D., Li, Y., Zhang, R., and Lin, L. (2016). Cost-effective active learning for deep image classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600.

Wang, S., Wang, Z., Che, W., Zhao, S., and Liu, T. (2021). Combining self-supervised learning and active learning for disfluency detection. *Transactions on Asian and Low-Resource Language Information Processing*, 21(3):1–25.

Weibull, W. (1951). A statistical distribution function of wide applicability. *Journal of applied mechanics*.

Weston, J., Chopra, S., and Bordes, A. (2015). Memory networks.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Yoo, D. and Kweon, I. S. (2019). Learning loss for active learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 93–102.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2018). Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*.

Yuan, M., Lin, H.-T., and Boyd-Graber, J. (2020). Cold-start active learning through self-supervised language modeling.

Zador, A. (2019). A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature Communications*, 10.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR.

Zenke, F., Poole, B., and Ganguli, S. (2017). Improved multitask learning through synaptic intelligence. *CoRR*, abs/1703.04200.

Zhu, J. and Bento, J. (2017). Generative adversarial active learning. *CoRR*, abs/1702.07956.

Zhu, J., Wang, H., Tsou, B. K., and Ma, M. (2009). Active learning with sampling by uncertainty and density for data annotations. *IEEE Transactions on audio, speech, and language processing*, 18(6):1323–1331.

Zhu, Y., Xu, W., Liu, Q., and Wu, S. (2020). When contrastive learning meets active learning: A novel graph active learning paradigm with self-supervision.