RICE UNIVERSITY

Theories and Perspectives on Practical Deep Learning

By

Cameron R. Wolfe

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE

Αναστασιος Κυριλλιδης
Αναστασιος Κυριλλιδης (May 31, 2023 13:52 CDT)

Anastasios Kyrillidis (Chair)

Noah Harding Assistant Professor, Computer Science

Christopher Jermaine
Christopher Jermaine (Jun 19, 2023 09:00 CDT)

Christopher Jermaine

Chair, Department of Computer Science
Director, Master of Data Science Program
Professor of Computer Science, J.S. Abercrombie Professor of Engineering

Santiago Segarra
Santiago Segarra (May 31, 2023 10:56 EDT)

Santiago Segarra

W. M. Rice Trustee Assistant Professor, Electrical and Computer Engineering

HOUSTON, TEXAS

May 2023

Theories and Perspectives on Practical Deep Learning

A dissertation submitted
in partial fulfillment
of requirements for
the degree of

DOCTOR OF PHILOSOPHY

by

CAMERON R. WOLFE
crw13@rice.edu

UNDER THE SUPERVISION OF

DR. ANASTASIOS KYRILLIDIS

DEPARTMENT OF COMPUTER SCIENCE
RICE UNIVERSITY

Contents

Abstract

Deep neural networks (DNNs) have proven to be adept at accurately automating many tasks (e.g., image and text classification [138, 145], object detection [203], text generation [198], and more). Across most domains, DNNs tend to achieve better performance with increasing scale, both in terms of dataset and model size [56, 116, 133]. As such, the benefit of DNNs comes at a steep computational (and monetary) cost, which can limit their applicability. This document aims to identify novel and intuitive techniques that can make deep learning more usable across domains and communities.

1 Overview

A goal of improving the overall usability of deep learning is broad and ambiguous. To make this goal more specific, we decompose deep learning workflows into two major components:

- The Model
- The Training Setup

All research presented in this document focuses upon one of these key areas. In the following sections, we will more rigorously define the goal of making deep learning more usable and highlight specific avenues by which this goal can be achieved. By connecting these ideas to the key components of a deep learning system outlined above, we identify specific, novel techniques that make leveraging deep learning-based technology much easier for everyday practitioners.

1.1 Motivation

What does it take for a technology like deep learning to become more widely adopted? Currently, three major considerations influence such adoption:

- *Performance*: the models and techniques we develop must solve their specified task accurately.
- *Cost*: the models and techniques we develop must be affordable.
- *Complexity*: the models and techniques we develop must be (relatively) simple to implement.

Any novel technique that claims to make deep learning more practically useful must tangibly improve (at least) one of these areas. In this document, we maintain this context and propose a variety of techniques that make the training and use of DNNs more performant, efficient, and accessible. Along the way, we place an emphasis upon methods that are both practical and provable (i.e., coming with theoretical guarantees).

1.2 The Model

One factor that influences the computational expense of deep learning is the size of current DNNs. Due to their size, DNNs are quite expensive to host and train, where we can define expense in terms of monetary cost, time to implement, training latency, required hardware, and more. Although large DNNs are incredibly powerful, their size creates a barrier to entry for their use—implementing systems and infrastructure to effectively support these models becomes increasingly difficult as they scale [45, 217]. Despite these considerations, DNNs seem to continue growing exponentially in size [27, 59]. With this in mind, we might wonder: *How can we make large models easier to use for "normal" practitioners?*

To reduce the memory overhead and latency of training large DNNs, we could completely avoid training the full model, adopting instead a sparse training strategy that separately trains small, randomly-sampled portions of the model. This approach was explored with the proposal of independent subnetwork training (IST) [258], which trains a feed-forward network by breaking it into several, smaller subnetworks that are trained in parallel on separate devices. Although this technique was later extended to other DNN architectures and training regimes [61, 62, 236], we consider its extension to graph convolutional networks (GCNs) [141] via Graph Independent Subnetwork Training (GIST) in Section 2. We find that performing training over a sparse selection of model parameters improves compute and memory efficiency, while maintaining or improving the performance of the underlying GCN. Overall, *such an approach enables larger GCNs to be trained to impressive performance levels with more modest resources.*

Beyond the difficulty of training large DNNs, deploying these models for use in practical applications can incur significant costs due to the hardware and compute requirements of hosting large models. Ideally, we would be able to capture the knowledge and performance of larger models with a much smaller DNN—*an idea that is being actively explored by neural network pruning techniques* [71, 163]. Although pruning methodologies for DNNs are mostly based upon heuristics [72, 110, 170], we explore pruning techniques that $i)$ perform well, $ii)$ come with theoretical guarantees, and $iii)$ run efficiently. For example, we propose an iterative, Sparse Structured Pruning algorithm (i-SpaSP) in Section 4 that runs orders of magnitude faster than other provable pruning techniques, performs competitively in experiments, and comes with theoretical performance guarantees for multi-layer DNNs. Additionally, in Section 3, we connect analysis of pruned DNNs [251] with existing convergence rates for DNNs trained with stochastic gradient descent [191], thus theoretically proving that high-performing subnetworks can be discovered with minimal pre-training. Going further, we extend this analysis to distributed pruning algorithms [19] that can be used to significantly speed up the pruning process.

Despite large models being favored in modern applications, we see in this document that leveraging sparsity in the DNN's underlying model architecture can make the training process more tractable with limited hardware resources and even improve DNN performance in the process. Going further, pruning techniques can be used to reduce the size of DNNs with minimal performance deterioration, which decouples the performance of such models from their size [74]. Here, we focus upon pruning techniques that come with rigorous theoretical guarantees and explore ideas that can improve the efficiency of such provable algorithms by orders of magnitude, thus making them more practically applicable. Combined together, the methods presented in this work focus upon modifications to the DNN model that, in turn, make the training and deployment of deep learning systems more performant and less costly.

1.3 The Training Setup

The standard training process for a DNN assumes access to a large database of training examples that may or may not have supervised labels. Using a pre-specified set of hyperparamters, the network then performs several training epochs over this dataset, having its parameters optimized according to some objective function. Although this strategy usually works well, there are a few notable issues that may arise when applied in the real world. For example, data is not static in most practical applications—*the training set changes constantly as new data becomes available*. Additionally, choosing suboptimal hyperparameters can drastically deteriorate training performance and efficiency, possibly even requiring the training process to be repeated from scratch. In this work, we explore solutions to such issues that can be easily implemented by practitioners, making DNNs more applicable and performant even in non-standard training settings.

Beyond being computationally expensive, the training process for DNNs requires the setting of several relevant hyperparameters, such as the learning rate or momentum. Currently, curated hyperparameter settings and schedules are widely-used across the deep learning community [37, 38]. If chosen correctly, hyperparameter schedules can improve DNN training efficiency and performance by a large margin [213, 216]. Furthermore, hyperparameter scheduling is a fundamental concept within deep learning that can be applied to a variety of different training settings [214, 243]. Inspired by prior work in low-precision training for DNNs [75, 76], we analyze a variety of different hyperparameter schedules that dynamically adapt network precision along the training trajectory in Section 6. Such scheduling approaches are simple to implement and have a huge impact on DNN training efficiency and performance.

Given that practical deep learning applications typically experience a constant influx of new data, DNNs trained using standard techniques must be intermittently re-trained to ensure available data is being modeled accurately. Re-training DNNs from scratch in this way is quite computationally expensive, thus raising the question: *Is there a training paradigm that can better cope with the dynamic nature of data?* Recently, several works have pioneered the training of DNNs in a streaming fashion [102, 103, 104]. With streaming, the learning process occurs in a single pass, where quick updates are performed over each new example of data that is exposed to the DNN. Although such techniques allow DNNs to be dynamically adapted to incoming streams of data, existing strategies require extensive pre-training procedures and leave a majority of network parameters fixed during streaming, which makes their implementation complex and can hinder performance. In Section 5, we explore a simplified, end-to-end streaming learning approach for DNNs, finding that such a technique is high-performing, cheap to deploy, simple to implement, and capable of being analyzed theoretically [29].

Although the standard approach for training DNNs is rarely questioned, we will see throughout the course of this document that exploring improvements to this process can yield impressive benefits. Streaming learning has the potential to revolutionize DNN training by allowing low cost adaptations of DNNs to new data, as opposed to constantly re-training them from scratch. By proposing simple, yet powerful, approaches for streaming learning, we enable practitioners to reduce the cost—in terms of implementation complexity, computational expense, latency, and more—of leveraging deep learning-based technology in highly dynamic environments. Additionally, exploring new avenues of hyperparameter scheduling (e.g., for DNN training precision) allows us to find small, practical modifications to the DNN training process that can improve the performance of DNNs while significantly reducing overhead.

2 GIST: Distributed Training for Large-Scale Graph Convolutional Networks

The graph convolutional network (GCN) is a go-to solution for machine learning on graphs, but its training is notoriously difficult to scale both in terms of graph size and the number of model parameters. Although some work has explored training on large-scale graphs, we pioneer efficient training of large-scale GCN models with the proposal of a novel, distributed training framework, called GIST. GIST disjointly partitions the parameters of a GCN model into several, smaller sub-GCNs that are trained independently and in parallel. Compatible with all GCN architectures and existing sampling techniques, GIST $i)$ improves model performance, $ii)$ scales to training on arbitrarily large graphs, $iii)$ decreases wall-clock training time, and $iv)$ enables the training of markedly overparameterized GCN models. Remarkably, with GIST, we train an astonishgly-wide

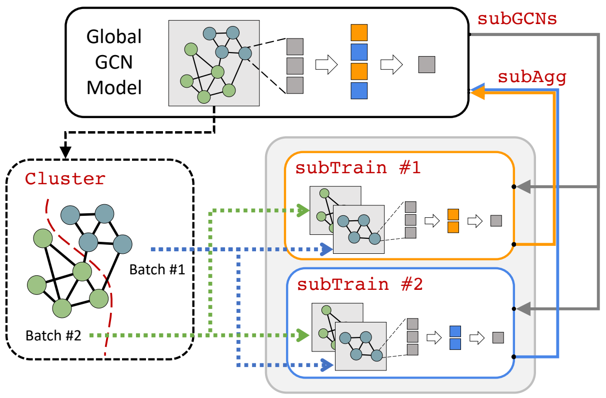

Figure 1: GIST pipeline: `subGCNs` divides the global GCN into sub-GCNs. Every sub-GCN is trained by `subTrain` using mini-batches (smaller sub-graphs) generated by `Cluster`. Sub-GCN parameters are intermittently aggregated through `subAgg`.

32 768-dimensional GraphSAGE model, which exceeds the capacity of a single GPU by a factor of $8\times$, to SOTA performance on the Amazon2M dataset.

2.1 Introduction

Since not all data can be represented in Euclidean space [26], many applications rely on graph-structured data. For example, social networks can be modeled as graphs by regarding each user as a node and friendship relations as edges [171, 186]. Alternatively, in chemistry, molecules can be modeled as graphs, with nodes representing atoms and edges encoding chemical bonds [14, 24].

To better understand graph-structured data, several (deep) learning techniques have been extended to the graph domain [50, 87, 174]. Currently, the most popular one is the graph convolutional network (GCN) [141], a multi-layer architecture that implements a generalization of the convolution operation to graphs. Although the GCN handles node—and graph—level classification, it is notoriously inefficient and unable to support large graphs [35, 36, 81, 122, 255, 262], making practical, large-scale applications difficult to handle.

To deal with these issues, node partitioning methodologies have been developed. These schemes can be roughly categorized into neighborhood sampling [36, 92, 276] and graph partitioning [43, 262] approaches. The goal is to partition a large graph into multiple smaller graphs that can be used as mini-batches for training the GCN. In this way, GCNs can handle larger graphs during training, expanding their potential into the realm of big data. However, the size of the underlying model is still limited by available memory capacity, thus placing further constraints on the scale of GCN experimentation.

Although some papers perform large-scale experiments [43, 262], the models (and data) used in GCN research remain small in the context of deep learning [141, 227], where the current trend is towards incredibly large models and datasets [27, 45]. Despite the widespread moral questioning of this trend [97, 169, 209], the deep learning community continues to push the limits of scale. Overparameterized models yield improvements in tasks like zero or few-shot learning [27, 199], are capable of discovering generalizable solutions [182], and even have desirable theoretical properties [191].

Although deeper GCNs may perform poorly due to oversmoothing [141, 149], GCNs should similarly benefit from overparameterization, meaning that larger hidden layers may be beneficial. Furthermore, recent work indicates that overparameterization is most impactful on larger datasets [116], making overparameterized models essential as GCNs are applied to practical problems at scale. Moving in this direction, *our work provides an efficient training framework for wide, overparameterized GCN models—beyond the memory capacity of a single GPU—of any architecture that is compatible with existing training techniques.*

Our Proposal. Inspired by independent subnetwork training (IST) [258], our methodology randomly partitions the hidden feature space in each layer, decomposing the global GCN model into multiple, narrow sub-GCNs of equal depth. Sub-GCNs are trained independently for several iterations in parallel prior to having their updates synchronized; see Figure 1. This process of randomly partitioning, independently training, and synchronizing sub-GCNs is repeated until convergence. We call this method **G**raph **I**ndependent **S**ubnetwork **T**raining (GIST), as it extends the IST framework to the training of GCNs.

Though IST was previously unexplored in this domain, we find that GIST pairs well with any GCN architecture, is compatible with node sampling techniques, can scale to arbitrarily large graphs, and significantly reduces wall-clock training time, allowing larger models and datasets to be explored. In particular, we focus on training "ultra-wide" GCNs (i.e., GCN models with very large hidden layers), as deeper GCNs are prone to oversmoothing [149] and GIST's model partitioning strategy can mitigate the memory overhead of training these wider GCNs. The contributions of this work are as follows:

- We develop a novel extension of IST for training GCNs, show that it works well for training GCNs with a variety of architectures, and demonstrate its compatibility with commonly-used GCN training techniques like neighborhood sampling and graph partitioning.
- We show that GIST can be used to reach state-of-the-art performance with reduced training time relative to standard training methodologies. GIST is a compatible addition to GCN training that improves efficiency.
- We use GIST to enable the training of markedly overparameterized GCN models. In particular, GIST is used to train a two-layer GraphSAGE model with a hidden dimension of 32 768 on the Amazon2M dataset. *Such a model exceeds the capacity of a single GPU by* $8\times$.

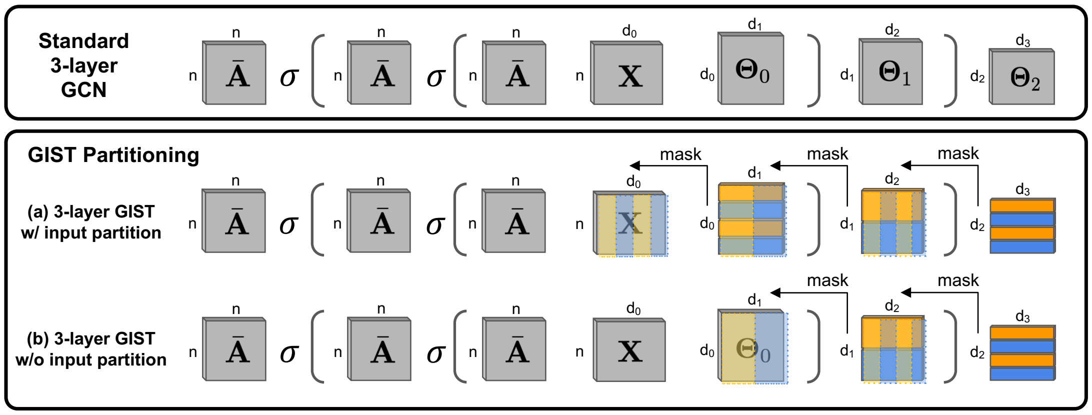
Standard 3-layer GCN
GIST Partitioning
(a) 3-layer GIST w/ input partition
(b) 3-layer GIST w/o input partition
mask
n
d0
d1
d2
d3
Ā
σ
X
Θ0
Θ1
Θ2

Figure 2: GCN partition into $m = 2$ sub-GCNs. Orange and blue colors depict different feature partitions. Both hidden dimensions (d_1 and d_2) are partitioned. The output dimension (d_3) is not partitioned. Partitioning the input dimension (d_0) is optional, but we do not partition d_0 in GIST.

2.2 What is the GIST of this work?

Algorithm 1: GIST Algorithm

Parameters: T synchronization iterations, m sub-GCNs, ζ local iterations, c clusters, $\mathcal{G}$ training graph.

```
$\Psi_{\mathcal{G}}(\cdot, \mathbf{\Theta}) \leftarrow$ randomly initialize GCN
$\{\mathcal{G}_{(j)}\}_{j=1}^{c} \leftarrow \texttt{Cluster}(\mathcal{G}, c)$
for $t = 0, \dots, T-1$ do
    $\left\{\Psi_{\mathcal{G}}(\cdot, \mathbf{\Theta}^{(i)})\right\}_{i=1}^{m} \leftarrow \texttt{subGCNs}(\Psi_{\mathcal{G}}(\cdot, \mathbf{\Theta}), m)$
    Distribute each $\Psi_{\mathcal{G}}(\cdot, \mathbf{\Theta}^{(i)})$ to a different worker
    for $i = 1, \dots, m$ do
        for $z = 1, \dots, \zeta$ do
            $\Psi_{\mathcal{G}}(\cdot, \mathbf{\Theta}^{(i)}) \leftarrow \texttt{subTrain}(\mathbf{\Theta}^{(i)}, \{\mathcal{G}_{(j)}\}_{j=1}^{c})$
        end
    end
    $\Psi_{\mathcal{G}}(\cdot, \mathbf{\Theta}) \leftarrow \texttt{subAgg}(\{\mathbf{\Theta}^{(i)}\}_{i=1}^{m})$
end
```

GCN Architecture. The GCN [141] is arguably the most widely-used neural network architecture on graphs. Consider a graph $\mathcal{G}$ comprised of n nodes with d-dimensional features $\mathbf{X} \in \mathbb{R}^{n \times d}$. The output $\mathbf{Y} \in \mathbb{R}^{n \times d'}$ of a GCN can be expressed as $\mathbf{Y} = \Psi_{\mathcal{G}}(\mathbf{X}, \mathbf{\Theta})$, where $\Psi_{\mathcal{G}}$ is an L-layered architecture with trainable parameters $\mathbf{\Theta}$. If we define $\mathbf{H}_0 = \mathbf{X}$, we then have that $\mathbf{Y} = \Psi_{\mathcal{G}}(\mathbf{X}, \mathbf{\Theta}) = \mathbf{H}_L$, where an intermediate ℓ-th layer of the GCN is given by

$$\mathbf{H}_{\ell+1} = \sigma(\bar{\mathbf{A}}\, \mathbf{H}_\ell\, \mathbf{\Theta}_\ell). \tag{1}$$

In (1), $\sigma(\cdot)$ is an element-wise activation function (e.g., ReLU), $\bar{\mathbf{A}}$ is the degree-normalized adjacency matrix of $\mathcal{G}$ with added self-loops, and the trainable parameters $\mathbf{\Theta} = \{\mathbf{\Theta}_\ell\}_{\ell=0}^{L-1}$ have dimensions $\mathbf{\Theta}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ with $d_0 = d$ and $d_L = d'$. In Figure 2 (top), we illustrate nested GCN layers for $L = 3$, but our methodology extends to arbitrary L. The activation function of the last layer is typically the identity or softmax transformation—we omit this in Figure 2 for simplicity.

GIST overview. We overview GIST in Algorithm 1 and present a schematic depiction in Figure 1. We partition our (randomly initialized) global GCN into m smaller, disjoint sub-GCNs with the `subGCNs` function ($m = 2$ in Figures 1 and 2) by sampling the feature space at each layer of the GCN; see Section 2.2.1. Each sub-GCN is assigned to a different worker (i.e., a different GPU) for ζ rounds of distributed, independent training through `subTrain`. Then, newly-learned sub-GCN parameters are aggregated (`subAgg`) into the global GCN model. This process repeats for T iterations. Our graph domain is partitioned into c subgraphs through the `Cluster` function ($c = 2$ in Figure 1). This operation is only relevant for large graphs ($n > 50\,000$), and we omit it ($c = 1$) for smaller graphs that don't require partitioning.[1]

2.2.1 `subGCNs`: Constructing Sub-GCNs

GIST partitions a global GCN model into several narrower sub-GCNs of equal depth. Formally, consider an arbitrary layer ℓ and a random, disjoint partition of the feature set $[d_\ell] = \{1, 2, \ldots, d_\ell\}$ into m equally-sized blocks $\{\mathcal{D}_\ell^{(i)}\}_{i=1}^m$.[2] Accordingly, we denote by $\mathbf{\Theta}_\ell^{(i)} = [\mathbf{\Theta}_\ell]_{\mathcal{D}_\ell^{(i)} \times \mathcal{D}_{\ell+1}^{(i)}}$ the matrix obtained by selecting from $\mathbf{\Theta}_\ell$ the rows and columns given by the ith blocks in the partitions of $[d_\ell]$ and $[d_{\ell+1}]$, respectively. With this notation in place, we can define m different sub-GCNs $\mathbf{Y}^{(i)} = \Psi_{\mathcal{G}}(\mathbf{X}^{(i)}; \mathbf{\Theta}^{(i)}) = \mathbf{H}_L^{(i)}$ where $\mathbf{H}_0^{(i)} = \mathbf{X}_{[n] \times \mathcal{D}_0^{(i)}}$ and each layer is given by:

$$\mathbf{H}_{\ell+1}^{(i)} = \sigma(\bar{\mathbf{A}}\, \mathbf{H}_\ell^{(i)}\, \mathbf{\Theta}_\ell^{(i)}). \tag{2}$$

Notably, not all parameters within the global GCN model are partitioned to a sub-GCN. However, by randomly re-constructing new groups of sub-GCNs according to a uniform distribution throughout the training process, all parameters have a high likelihood of being updated.

Sub-GCN partitioning is illustrated in Figure 2-(a), where $m = 2$. Partitioning the input features is optional (i.e., (a) vs. (b) in Figure 2). *We do not partition the input features within GIST* so that sub-GCNs have identical input information (i.e., $\mathbf{X}^{(i)} = \mathbf{X}$ for all i); see Section 2.4.1. Similarly, we do not partition the output feature space to ensure that the sub-GCN output dimension coincides with that of the global model, thus avoiding any need to modify the loss function. This decomposition procedure (`subGCNs` function in Algorithm 1) extends to arbitrary L.

2.2.2 `subTrain`: Independently Training Sub-GCNs

Assume $c = 1$ so that the `Cluster` operation in Algorithm 1 is moot and $\{\mathcal{G}_{(j)}\}_{j=1}^c = \mathcal{G}$. Because $\mathbf{Y}^{(i)}$ and $\mathbf{Y}$ share the same dimension, sub-GCNs can be trained to minimize the same global loss function. One

[1]Though any clustering method can be used, we advocate the use of METIS [134, 135] due to its proven efficiency in large-scale graphs.

[2]For example, if $d_\ell = 4$ and $m = 2$, one valid partition would be given by $\mathcal{D}_\ell^{(1)} = \{1, 4\}$ and $\mathcal{D}_\ell^{(2)} = \{2, 3\}$.

application of `subTrain` in Algorithm 1 corresponds to a single step of stochastic gradient descent (SGD). Inspired by local SGD [158], multiple, independent applications of `subTrain` are performed in parallel (i.e., on separate GPUs) for each sub-GCN prior to aggregating weight updates. The number of independent training iterations between synchronization rounds, referred to as *local iterations*, is denoted by ζ, and the total amount of training is split across sub-GCNs.[3] Ideally, the number sub-GCNs and local iterations should be increased as much as possible to minimize communication and training costs. In practice, however, such benefits may come at the cost of statistical inefficiency; see Section 2.4.1.

If $c > 1$, `subTrain` first selects one of the c subgraphs in $\{\mathcal{G}_{(j)}\}_{j=1}^{c}$ to use as a mini-batch for SGD. Alternatively, the union of several sub-graphs in $\{\mathcal{G}_{(j)}\}_{j=1}^{c}$ can be used as a mini-batch for training. Aside from using mini-batches for each SGD update instead of the full graph, *the use of graph partitioning does not modify the training approach outlined above.* Some form of node sampling must be adopted to make training tractable when the full graph is too large to fit into memory. However, *both graph partitioning and layer sampling are compatible with* GIST (see Sections 2.4.2 and 2.4.4). We adopt graph partitioning in the main experiments due to the ease of implementation. The novelty of our work lies in the feature partitioning strategy of GIST for distributed training, which is an orthogonal technique to node sampling; see Figure 3 and Section 2.2.4.

2.2.3 `subAgg`: Aggregating Sub-GCN Parameters

After each sub-GCN completes ζ training iterations, their updates are aggregated into the global model (`subAgg` function in Algorithm 1). Within `subAgg`, each worker replaces global parameter entries within $\mathbf{\Theta}$ with its own, independently-trained sub-GCN parameters $\mathbf{\Theta}^{(i)}$, where no collisions occur due to the disjointness of sub-GCN partitions. Thus, `subAgg` is a basic copy operation that transfers sub-GCN parameters into the global model.

Not every parameter in the global GCN model is updated by `subAgg` because, as previously mentioned, parameters exist that are not partitioned to any sub-GCN by the `subGCNs` operation. For example, focusing on $\mathbf{\Theta}_1$ in Figure 2-(a), one worker will be assigned $\mathbf{\Theta}_1^{(1)}$ (i.e., overlapping orange blocks), while the other worker will be assigned $\mathbf{\Theta}_1^{(2)}$ (i.e., overlapping blue blocks). The rest of $\mathbf{\Theta}_1$ is not considered within `subAgg`. Nonetheless, since sub-GCN partitions are randomly drawn in each cycle t, one expects all of $\mathbf{\Theta}$ to be updated multiple times if T is sufficiently large.

2.2.4 What is the value of GIST?

Architecture-Agnostic Distributed Training. GIST is a generic, distributed training methodology that can be used for any GCN architecture. We implement GIST for vanilla GCN, GraphSAGE, and GAT architectures, but GIST is not limited to these models; see Section 2.4.

Compatibility with Sampling Methods. GIST is **<u>NOT</u>** a replacement for graph or layer sampling. Rather, it is an efficient, distributed training technique that can be used in tandem with node partitioning. As depicted in

[3]For example, if a global model is trained on a single GPU for 10 epochs, a comparable experiment for GIST with two sub-GCNs would train each sub-GCN for only 5 epochs.

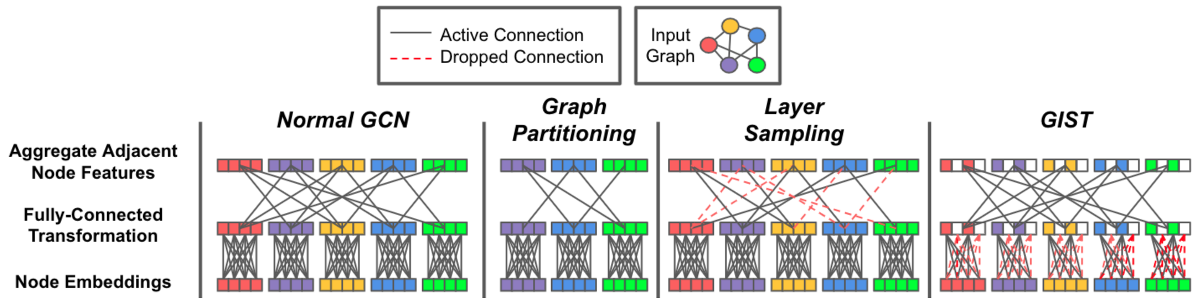

Figure 3: Illustrates the difference between GIST and node sampling techniques within the forward pass of a single GCN layer (excluding non-linear activation). While graph partitioning and layer sampling remove nodes from the forward pass (i.e., either completely or on a per-layer basis), GIST partitions node feature representations (and, in turn, model parameters) instead of the nodes themselves.

Figure 3, GIST partitions node feature representations and model parameters between sub-GCNs, while graph partitioning and layer sampling sub-sample nodes within the graph.

Interestingly, we find that GIST's feature and parameter partitioning strategy is compatible with node partitioning—the two approaches can be combined to yield further efficiency benefits. For example, GIST is combined with graph partitioning strategies in Section 2.4.2 and with layer sampling methodologies in Section 2.4.4. As such, we argue that GIST offers an easy add-on to GCN training that makes larger scale experiments more feasible.

Enabling Ultra-Wide GCN Training. GIST indirectly updates the global GCN through the training of smaller sub-GCNs, enabling models with hidden dimensions that exceed the capacity of a single GPU to be trained; in our experiments, we show results where GIST allows training of models beyond the capacity of a single GPU by a factor of $8\times$. In this way, GIST allows markedly overparametrized ("ultra-wide") GCN models to be trained on existing hardware. In Section 2.4.2, *we leverage this capability to train a two-layer GCN model with a hidden dimension of* 32 768 *on Amazon2M*. Overparameterization through width is especially relevant to GCNs because deeper models suffer from oversmoothing [149]. We do not explore depth-wise partitions of different GCN layers to each worker, but rather focus solely upon partitioning the hidden neurons within each layer.

Improved Model Complexity. Consider a single GCN layer, trained over M machines with input and output dimension of d_{i-1} and d_i, respectively. For one synchronization round, the communication complexity of GIST and standard distributed training is $\mathcal{O}(\frac{1}{M}d_id_{i-1})$ and $\mathcal{O}(Md_id_{i-1})$, respectively. GIST reduces communication by only communicating sub-GCN parameters. Existing node partitioning techniques cannot similarly reduce communication complexity because model parameters are never partitioned. Furthermore, the computational complexity of the forward pass for a GCN model trained with GIST and using standard methodology is $\mathcal{O}(\frac{1}{M}N^2d_i + \frac{1}{M^2}Nd_id_{i-1})$ and $\mathcal{O}(N^2d_i + Nd_id_{i-1})$, respectively, where N is the number of nodes in the partition being processed.[4] Node partitioning can reduce N by a constant factor but is compatible with GIST.

Relation to IST. Our work extends the IST distributed training framework—originally proposed for fully-

[4]We omit the complexity of applying the element-wise activation function for simplicity.

connected network architectures [258]—to GCNs. Due to the unique aspects of GCN training (e.g., non-euclidean data and aggregation of node features), it was previously unclear whether IST would work well in this domain. Though IST is applicable to a variety of architectures, we find that it is especially useful for efficiently training GCNs to high accuracy. GIST $i)$ provides speedups and performance benefits, $ii)$ is compatible with other efficient GCN training methods, and $iii)$ enables training of uncharacteristically-wide GCN models, allowing overparameterized GCNs to be explored via greater width. The practical utility of GIST and interplay of the approach with unique aspects of GCN training differentiate our work from the original IST proposal.

2.3 Related Work

GCN training. In spite of their widespread success in several graph related tasks, GCNs often suffer from training inefficiencies[81, 122]. Consequently, the research community has focused on developing efficient and scalable algorithms for training GCNs [35, 36, 43, 92, 262, 276]. The resulting approaches can be divided roughly into two areas: *neighborhood sampling* and *graph partitioning*. However, it is important to note that these two broad classes of solutions are not mutually exclusive, and reasonable combinations of the two approaches may be beneficial.

Neighborhood sampling methodologies aim to sub-select neighboring nodes at each layer of the GCN, thus limiting the number of node representations in the forward pass and mitigating the exponential expansion of the GCNs receptive field. VRGCN [35] implements a variance reduction technique to reduce the sample size in each layer, which achieves good performance with smaller graphs. However, it requires to store all the intermediate node embeddings during training, leading to a memory complexity close to full-batch training. GraphSAGE [92] learns a set of aggregator functions to gather information from a node's local neighborhood. It then concatenates the outputs of these aggregation functions with each node's own representation at each step of the forward pass. FastGCN [36] adopts a Monte Carlo approach to evaluate the GCN's forward pass in practice, which computes each node's hidden representation using a fixed-size, randomly-sampled set of nodes. LADIES [276] introduces a layer-conditional approach for node sampling, which encourages node connectivity between layers in contrast to FastGCN [36].

Graph partitioning schemes aim to select densely-connected sub-graphs within the training graph, which can be used to form mini-batches during GCN training. Such sub-graph sampling reduces the memory footprint of GCN training, thus allowing larger models to be trained over graphs with many nodes. ClusterGCN [43] produces a very large number of clusters from the global graph, then randomly samples a subset of these clusters and computes their union to form each sub-graph or mini-batch. Similarly, GraphSAINT [262] randomly samples a sub-graph during each GCN forward pass. However, GraphSAINT also considers the bias created by unequal node sampling probabilities during sub-graph construction, and proposes normalization techniques to eliminate this bias.

As explained in Section 2.2, GIST also relies on graph partitioning techniques (`Cluster`) to handle large graphs. However, the feature sampling scheme at each layer (`subGCNs`) that leads to parallel and narrower sub-GCNs is a hitherto unexplored framework for efficient GCN training.

Distributed training. Distributed training is a heavily studied topic [210, 268]. Our work focuses on

m	d_0	d_1	d_2	Cora	Citeseer	Pubmed	OGBN-Arxiv
Baseline				81.52 ± 0.005	75.02 ± 0.018	75.90 ± 0.003	70.85 ± 0.089
2	✓	✓	✓	80.00 ± 0.010	$\mathbf{75.95} \pm 0.007$	76.68 ± 0.011	65.65 ± 0.700
	✓	✓		78.30 ± 0.011	69.34 ± 0.018	75.78 ± 0.015	65.33 ± 0.347
		✓	✓	$\mathbf{80.82} \pm 0.010$	75.82 ± 0.008	$\mathbf{78.02} \pm 0.007$	$\mathbf{70.10} \pm 0.224$
4	✓	✓	✓	76.78 ± 0.017	70.66 ± 0.011	65.67 ± 0.044	54.21 ± 1.360
	✓	✓		66.56 ± 0.061	68.38 ± 0.018	68.44 ± 0.014	52.64 ± 1.988
		✓	✓	$\mathbf{81.18} \pm 0.007$	$\mathbf{76.21} \pm 0.017$	$\mathbf{76.99} \pm 0.006$	$\mathbf{68.69} \pm 0.579$
8	✓	✓	✓	48.32 ± 0.087	45.42 ± 0.092	54.29 ± 0.029	40.26 ± 1.960
	✓	✓		53.60 ± 0.020	54.68 ± 0.030	51.44 ± 0.002	26.84 ± 7.226
		✓	✓	$\mathbf{79.58} \pm 0.006$	$\mathbf{75.39} \pm 0.016$	$\mathbf{76.99} \pm 0.006$	$\mathbf{65.81} \pm 0.378$

Table 1: Test accuracy of GCN models trained on small-scale datasets with GIST. We selectively partition each feature dimension within the GCN model, indicated by a check mark.

synchronous and distributed training techniques [154, 256, 266]. Some examples of synchronous, distributed training approaches include data parallel training, parallel SGD [4, 274], and local SGD [158, 221]. Our methodology holds similarities to model parallel training techniques, which have been heavily explored [23, 83, 89, 142, 194, 272]. More closely, our approach is inspired by IST, explored for feed-forward networks in [258]. Later work analyzed IST theoretically [155] and extended its use to more complex ResNet architectures [62]. We empirically explore the extension of IST to the GCN architecture, finding that IST-based methods are suited well for GCN training. However, the IST framework is applicable to network architectures beyond the GCN.

2.4 Experiments

We use GIST to train different GCN architectures on six public, multi-node classification datasets; see Appendix A.1 for details. In most cases, we compare the performance of models trained with GIST to that of models trained with standard methods (i.e., single GPU with node partitioning). Comparisons to models trained with other distributed methodologies are also provided in Appendix A.2. Experiments are divided into small and large scale regimes based upon graph size. The goal of GIST is to $i)$ train GCN models to state-of-the-art performance, $ii)$ minimize wall-clock training time, and $iii)$ enable training of very wide GCN models.

2.4.1 Small-Scale Experiments

In this section, we perform experiments over Cora, Citeseer, Pubmed, and OGBN-Arxiv datasets [120, 208]. For these small-scale datasets, we train a three-layer, 256-dimensional GCN model [141] with GIST; see Appendix A.1.3 for further experimental settings. All reported metrics are averaged across five separate trials. Because these experiments run quickly, we use them to analyze the impact of different design and hyperparameter choices rather than attempting to improve runtime (i.e., speeding up such short experiments is futile).

		Reddit Dataset					
L	m	GraphSAGE			GAT		
		F1	Time	Speedup	F1	Time	Speedup
2	-	96.09	105.78s	1.00×	89.57	1.19hr	1.00×
	2	96.40	70.29s	1.50×	90.28	0.58hr	2.05×
	4	96.16	68.88s	1.54×	90.02	0.31hr	3.86×
	8	95.46	76.68s	1.38×	89.01	0.18hr	6.70×
3	-	96.32	118.37s	1.00×	89.25	2.01hr	1.00×
	2	96.36	80.46s	1.47×	89.63	0.95hr	2.11×
	4	95.76	78.74s	1.50×	88.82	0.48hr	4.19×
	8	94.39	88.54s	(1.34×)	70.38	0.26hr	(7.67×)
4	-	96.32	120.74s	1.00×	88.36	2.77hr	1.00×
	2	96.01	91.75s	1.32×	87.97	1.31hr	2.11×
	4	95.21	78.74s	(1.53×)	78.42	0.66hr	(4.21×)
	8	92.75	88.71s	(1.36×)	66.30	0.35hr	(7.90×)

		Amazon2M Dataset					
L	m	GraphSAGE ($d_i = 400$)			GraphSAGE ($d_i = 4\,096$)		
		F1	Time	Speedup	F1	Time	Speedup
2	-	89.90	1.81hr	1.00×	91.25	5.17hr	1.00×
	2	88.36	1.25hr	(1.45×)	90.70	1.70hr	3.05×
	4	86.33	1.11hr	(1.63×)	89.49	1.13hr	(4.57×)
	8	84.73	1.13hr	(1.61×)	88.86	1.11hr	(4.65×)
3	-	90.36	2.32hr	1.00×	91.51	9.52hr	1.00×
	2	88.59	1.56hr	(1.49×)	91.12	2.12hr	4.49×
	4	86.46	1.37hr	(1.70×)	89.21	1.42hr	(6.72×)
	8	84.76	1.37hr	(1.69×)	86.97	1.34hr	(7.12×)
4	-	90.40	3.00hr	1.00×	91.61	14.20hr	1.00×
	2	88.56	1.79hr	(1.68×)	91.02	2.77hr	5.13×
	4	87.53	1.58hr	(1.90×)	89.07	1.65hr	(8.58×)
	8	85.32	1.56hr	(1.93×)	87.53	1.55hr	(9.13×)

Table 2: Performance of models trained with GIST on Reddit and Amazon2M. Parenthesis are placed around speedups achieved at a cost of >1 deterioration in F1 and $m =$"-" refers to the baseline.

Which layers should be partitioned? We investigate whether models trained with GIST are sensitive to the partitioning of features within certain layers. Although the output dimension d_3 is never partitioned, we selectively partition dimensions d_0, d_1, and d_2 to observe the impact on model performance; see Table 1. Partitioning input features (d_0) significantly degrades test accuracy because sub-GCNs observe only a portion of each node's input features (i.e., this becomes more noticeable with larger m). However, other feature dimensions cause no performance deterioration when partitioned between sub-GCNs, *leading us to partition all feature dimensions other than* d_0 *and* d_L within the final GIST methodology; see Figure 2-(b).

How many Sub-GCNs to use? Using more sub-GCNs during GIST training typically improves runtime because sub-GCNs $i)$ become smaller, $ii)$ are each trained for fewer epochs, and $iii)$ are trained in parallel. We find that *all models trained with GIST perform similarly for practical settings of* m; see Table 1. One may continue increasing the number sub-GCNs used within GIST until all GPUs are occupied or model performance begins to decrease.

GIST Performance. Models trained with GIST often exceed the performance of models trained with standard, single-GPU methodology; see Table 1. Intuitively, we hypothesize that the random feature partitioning within GIST, which loosely resembles dropout [220], provides regularization benefits during training.

2.4.2 Large-Scale Experiments

For large-scale experiments on Reddit and Amazon2M, the baseline model is trained on a single GPU and compared to models trained with GIST in terms of F1 score and training time. All large-scale graphs are partitioned into 15 000 sub-graphs during training.[5] Graph partitioning is mandatory because the training graphs are too large to fit into memory. One could instead use layer sampling to make training tractable (see Section 2.4.4), but we adopt graph partitioning in most experiments because the implementation is simple and performs well.

Reddit Dataset. We perform tests with 256-dimensional GraphSAGE [92] and GAT [227] models with two to four layers on Reddit; see Appendix A.1.4 for more details. As shown in Table 2, utilizing GIST significantly accelerates GCN training (i.e., a $1.32\times$ to $7.90\times$ speedup). GIST performs best in terms of F1 score with $m = 2$ sub-GCNs (i.e., $m = 4$ yields further speedups but F1 score decreases). Interestingly, *the speedup provided by GIST is more significant for models and datasets with larger compute requirements*. For example, experiments with the GAT architecture, which is more computationally expensive than GraphSAGE, achieve a near-linear speedup with respect to m.

Amazon2M Dataset. Experiments are performed with two, three, and four-layer GraphSAGE models [92] with hidden dimensions of 400 and $4\,096$ (we refer to these models as “narrow” and “wide”, respectively). We compare the performance (i.e., F1 score and wall-clock training time) of GCN models trained with standard, single-GPU methodology to that of models trained with GIST; see Table 2. Narrow models trained with GIST have a lower F1 score in comparison to the baseline, but training time is significantly reduced. For wider models, GIST provides a more significant speedup (i.e., up to $7.12\times$) and tends to achieve comparable F1 score in comparison to the baseline, revealing that *GIST works best with wider models*.

Within Table 2, models trained with GIST tend to achieve a wall-clock speedup at the cost of a lower F1 score (i.e., observe the speedups marked with parenthesis in Table 2). When training time is analyzed with respect to a fixed F1 score, we observe that the baseline takes significantly longer than GIST to achieve a fixed F1 score. For example, when $L = 2$, a wide GCN trained with GIST ($m = 8$) reaches an F1 score of 88.86 in $\sim$ 4 000 seconds, *while models trained with standard methodology take* $\sim$ 10 000 *seconds to achieve a comparable F1 score.* As such, GIST significantly accelerates training relative to model performance.

[5]Single-GPU training with graph partitioning via METIS is the same approach adopted by ClusterGCN [43], making our single-GPU baseline a ClusterGCN model. We adopt the same number of sub-graphs as proposed in this work.

L	m	F1 Score (Time in hours)				
		$d_i = 400$	$d_i = 4\ 096$	$d_i = 8\ 192$	$d_i = 16\ 384$	$d_i = 32\ 768$
2	-	89.38 (1.81)	90.58 (5.17)	OOM	OOM	OOM
	2	87.48 (1.25)	90.09 (1.70)	90.87 (2.76)	90.94 (9.31)	90.91 (32.31)
	4	84.82 (1.11)	88.79 (1.13)	89.76 (1.49)	90.10 (2.24)	90.17 (5.16)
	8	82.56 (1.13)	87.16 (1.11)	88.31 (1.20)	88.89 (1.39)	89.46 (1.76)
3	-	89.73 (2.32)	90.99 (9.52)	OOM	OOM	OOM
	2	87.79 (1.56)	90.40 (2.12)	90.91 (4.87)	91.05 (17.7)	OOM
	4	85.30 (1.37)	88.51 (1.42)	89.75 (2.07)	90.15 (3.44)	OOM
	8	82.84 (1.37)	86.12 (1.34)	88.38 (1.37)	88.67 (1.88)	88.66 (2.56)
4	-	89.77 (3.00)	91.02 (14.20)	OOM	OOM	OOM
	2	87.75 (1.79)	90.36 (2.77)	91.08 (6.92)	91.09 (26.44)	OOM
	4	85.32 (1.58)	88.50 (1.65)	89.76 (2.36)	90.05 (4.93)	OOM
	8	83.45 (1.56)	86.60 (1.55)	88.13 (1.61)	88.44 (2.30)	OOM

Table 3: Performance of GraphSAGE models of different widths trained with GIST on Amazon2M. $m =$"-" refers to the baseline and "OOM" marks experiments that cause out-of-memory errors.

2.4.3 Training Ultra-Wide GCNs

We use GIST to train GraphSAGE models with widths as high as 32 000 (i.e., $\mathbf{8}\times$ beyond the capacity of a single GPU); see Table 3 for results and Appendix A.1.5 for more details. Considering $L = 2$, the best-performing, single-GPU GraphSAGE model ($d_i = 4\ 096$) achieves an F1 score of 90.58 in 5.2 hours. With GIST ($m = 2$), we achieve a higher F1 score of 90.87 in 2.8 hours (i.e., a $1.86\times$ speedup) using $d_i = 8\ 192$, which is beyond single GPU capacity. Similar patterns are observed for deeper models. Furthermore, we find that utilizing larger hidden dimensions yields further performance improvements, revealing the utility of wide, overparameterized GCN models. *GIST, due to its feature partitioning strategy, is unique in its ability to train models of such scale to state-of-the-art performance*.

2.4.4 GIST with Layer Sampling

As previously mentioned, some node partitioning approach must be adopted to avoid memory overflow when the underlying training graph is large. Although graph partitioning is used within most experiments (see Section 2.4.2), GIST is also compatible with other node partitioning strategies. To demonstrate this, we perform training on Reddit using GIST combined with a recent layer sampling approach [276] (i.e., instead of graph partitioning); see Appendix A.1.6 for more details.

As shown in Table 4, combining GIST with layer sampling enables training on large-scale graphs, and the observed speedup actually exceeds that of GIST with graph partitioning. For example, GIST with layer sampling yields a $1.83\times$ speedup when $L = 2$ and $m = 2$, in comparison to a $1.50\times$ speedup when graph partitioning is used within GIST (see Table 2). As the number of sub-GCNs is increased beyond $m = 2$, GIST with layer sampling continues to achieve improvements in wall-clock training time (e.g., speedup increases from $1.83\times$ to $2.90\times$ from $m = 2$ to $m = 4$ for $L = 2$) without significant deterioration to model

L	# Sub-GCNs	GIST + LADIES		
		F1 Score	Time	Speedup
2	Baseline	89.73	3 359.91s	1.00×
	2	89.29	1 834.59s	1.83×
	4	88.42	1 158.51s	2.90×
3	Baseline	89.57	4 803.88s	1.00×
	2	86.52	2 635.18s	1.82×
	4	86.72	1 605.32s	3.00×

Table 4: Performance of GCN models trained with a combination of GIST and LADIES [276] on Reddit. Here, the baseline represents models trained with LADIES in a standard, single-GPU manner.

performance. Thus, although node partitioning is needed to enable training on large-scale graphs, the feature partitioning strategy of GIST is compatible with numerous sampling strategies (i.e., not just graph sampling).

2.5 Conclusion

We present GIST, a distributed training approach for GCNs that enables the exploration of larger models and datasets. GIST is compatible with existing sampling approaches and leverages a feature-wise partition of model parameters to construct smaller sub-GCNs that are trained independently and in parallel. We have shown that GIST achieves remarkable speed-ups over large graph datasets and even enables the training of GCN models of unprecedented size. We hope GIST can empower the exploration of larger, more powerful GCN architectures within the graph community.

3 How much pre-training is enough to discover a good subnetwork?

Neural network pruning is useful for discovering efficient, high-performing subnetworks within pre-trained, dense network architectures. However, more often than not, it involves a three-step process—pre-training, pruning, and re-training—that is computationally expensive, as the dense model must be fully pre-trained. Luckily, several works have empirically shown that high-performing subnetworks can be discovered via pruning without fully pre-training the dense network. Aiming to theoretically analyze the amount of dense network pre-training needed for a pruned network to perform well, we discover a theoretical bound in the number of SGD pre-training iterations on a two-layer, fully-connected network, beyond which pruning via greedy forward selection [251] yields a subnetwork that achieves good training error. This threshold is shown to be logarithmically dependent upon the size of the dataset, meaning that experiments with larger datasets require more pre-training for subnetworks obtained via pruning to perform well. Additionally, we propose a distributed version of greedy forward selection that significantly speeds up the pruning process via parallelization across several compute sites and yields identical theoretical guarantees. We empirically demonstrate the validity of our theoretical results across a variety of architectures and datasets, including fully-connected networks trained on MNIST and several deep convolutional neural network (CNN) architectures trained on CIFAR10 and ImageNet.

3.1 Introduction

The proposal of the Lottery Ticket Hypothesis (LTH) [71] has led to significant interest in using pruning techniques to discover small (sometimes sparse) models that perform well. LTH has been empirically validated and is even applicable in large-scale settings [39, 72, 79, 163, 180, 269, 271], revealing that high-performing subnetworks can be obtained by pruning dense, pre-trained models and fine-tuning the pruned weights to convergence (i.e., either from their initial values or some later point) [40, 84, 204].

Neural network pruning tends to follow a three step process, including pre-training, pruning, and re-training, where pre-training is the most costly step [71, 254]. To circumvent the cost of pre-training, several works explore the possibility of pruning networks directly from initialization (i.e., the "strong lottery ticket hypothesis") [73, 200, 232], but subnetwork performance could suffer. Adopting a hybrid approach, good subnetworks can also be obtained from models with minimal pre-training [41, 254] (i.e., "early-bird" tickets), providing hope that high-performing pruned models can be discovered without incurring the full training cost of the dense model.

Empirical analysis of pruning techniques has inspired associated theoretical developments. Several works have derived bounds for the performance and size of subnetworks discovered in randomly-initialized networks [173, 190, 195]. Other theoretical works analyze pruning via greedy forward selection [251, 252]. In addition to enabling analysis with respect to subnetwork size, pruning via greedy forward selection was shown to work well in practice for large-scale architectures and datasets. However, greedy forward selection is quite slow compared to heuristic pruning techniques (e.g., pruning based on ℓ_1-norm of weights [148]), which diminishes the algorithm's practical applicability. Some findings from these works apply to randomly-initialized networks given proper assumptions [173, 190, 195, 251], *but no work yet analyzes how different levels of pre-training impact the performance of pruned networks from a theoretical perspective.*

Our Proposal. We adopt the greedy forward selection pruning framework and analyze subnetwork performance with respect to the number of SGD pre-training iterations performed on the dense model. From this analysis, *we discover a threshold in the number of pre-training iterations—logarithmically dependent upon the size of the dataset—beyond which subnetworks obtained via greedy forward selection perform well in terms of training error.* Such a finding offers theoretical insight into the early-bird ticket phenomenon and provides intuition for why discovering high-performing subnetworks is more difficult in large-scale experiments [163, 204, 254].

Given that pruning via greedy forward selection is slow compared to heuristic techniques, we propose a distributed variant of greedy forward selection that can parallelize and accelerate the pruning process across multiple compute sites. Distributed greedy forward selection is shown to achieve identical theoretical guarantees compared to the centralized variant and used to accelerate experiments with greedy forward selection within this work. In particular, we perform extensive experiments (i.e., two-layer networks on MNIST and CNNs on CIFAR10 and ImageNet) to validate our theoretical analysis, *finding that the amount of pre-training required to discover a subnetwork that performs well is consistently dependent on the size of the dataset.*

3.2 Preliminaries

Notation. Vectors are represented with bold type (e.g., $\mathbf{x}$), while scalars are represented by normal type (e.g., x). $\|\cdot\|_2$ represents the ℓ_2 vector norm. Unless otherwise specified, vector norms are always considered to be ℓ_2 norms. $[N]$ is used to represent the set of positive integers from 1 to N (i.e., $[N] = \{1 \dots N\}$). We denote the ball of radius r centered at $\boldsymbol{\mu}$ as $\mathcal{B}(\boldsymbol{\mu}, r)$.

Network Parameterization. We consider two-layer neural networks of width N:

$$f(\mathbf{x}, \boldsymbol{\Theta}) = \tfrac{1}{N} \sum_{i=1}^{N} \sigma(\mathbf{x}, \boldsymbol{\theta}_i), \tag{3}$$

where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N\}$ represents all weights within the two-layer neural network. In (3), $\sigma(\cdot, \boldsymbol{\theta}_i)$ represents the activation of a single neuron as shown below:

$$\sigma(\mathbf{x}, \boldsymbol{\theta}_i) = b_i \sigma_+(\mathbf{a}_i^\top \mathbf{x}). \tag{4}$$

Here, σ_+ is an activation function with bounded 1st and 2nd-order derivatives; see Assumption 1. The weights of a single neuron are $\boldsymbol{\theta}_i = [b_i, \mathbf{a}_i] \in \mathbb{R}^{d+1}$, where d is the dimension of the input vector.

Two-layer networks are relevant to larger-scale architectures and applications [18, 113, 189] and have the special property of separability between hidden neurons—the network's output can be derived by computing (4) separately for each neuron. Notably, we do not leverage any results from mean-field analysis of two-layer networks [219, 251], choosing instead to analyze the network's performance when trained directly with stochastic gradient descent (SGD) [191].

The Dataset. We assume that our network is modeling a dataset D, where $|D| = m$. $D = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}_{i=1}^{m}$, where $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ for all $i \in [m]$. During training, we consider an ℓ_2-norm regression loss over the dataset:

$$\mathcal{L}[f] = \tfrac{1}{2} \cdot \mathbb{E}_{(\mathbf{x},y)\sim D} \left[\left(f(\mathbf{x}, \boldsymbol{\Theta}) - y\right)^2 \right]. \tag{5}$$

We define $\mathbf{y} = \left[y^{(1)}, y^{(2)}, \dots, y^{(m)}\right] / \sqrt{m}$, which represents a concatenated vector of all labels within the dataset scaled by a factor $\sqrt{m}$. Similarly, we define $\phi_{i,j} = \sigma(\mathbf{x}^{(j)}, \boldsymbol{\theta}_i)$ as the output of neuron i for the j-th input vector in the dataset and construct the vector $\boldsymbol{\Phi}_i = [\phi_{i,1}, \phi_{i,2}, \dots, \phi_{i,m}] / \sqrt{m}$, which is a scaled, concatenated vector of output activations for a single neuron across the entire dataset. We use $\mathcal{M}_N$ to denote the convex hull over such activation vectors for all N neurons:

$$\mathcal{M}_N = \texttt{Conv}\left\{\boldsymbol{\Phi}_i : i \in [N]\right\}. \tag{6}$$

Here, $\texttt{Conv}\{\cdot\}$ denotes the convex hull. $\mathcal{M}_N$ forms a marginal polytope of the feature map for all neurons in the two-layer network across every dataset example. We use $\texttt{Vert}(\mathcal{M}_N)$ to denote the vertices of the marginal polytope $\mathcal{M}_N$ (i.e., $\texttt{Vert}(\mathcal{M}_N) = \{\boldsymbol{\Phi}_i : i \in [N]\}$). Using the construction $\mathcal{M}_N$, the ℓ_2 loss can be easily defined as follows:

$$\ell(\mathbf{z}) = \frac{1}{2}\|\mathbf{z} - \mathbf{y}\|^2, \tag{7}$$

where $\mathbf{z} \in \mathcal{M}_N$. Finally, we define the diameter of the space $\mathcal{M}_N$ as $\mathcal{D}_{\mathcal{M}_N} = \max_{\mathbf{u},\mathbf{v} \in \mathcal{M}_N} \|\mathbf{u} - \mathbf{v}\|_2$.

Distributed Pruning. For the distributed variant of greedy forward selection, we consider local compute nodes $\mathcal{V} = \{v_i\}_{i=1}^V$, which communicate according to an undirected connected graph $G = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{E}$ is a set of edges, where $|\mathcal{E}| = E$ and $(v_i, v_j) \in \mathcal{E}$ indicates that nodes v_i and v_j can communicate with each other. For simplicity, our analysis assumes synchronous updates, that the network has no latency, and that each node has an identical copy of the data D.

3.3 Pruning with greedy forward selection

Algorithm 2: Centralized Greedy Forward Selection for Two-Layer Networks

```
$\mathbf{z}_0 = \mathbf{0}$
for $k = 1, 2, \dots$ do
    # Step I: select a new neuron
    $\mathbf{q}_k = \arg\min_{\mathbf{q} \in \mathtt{Vert}(\mathcal{M}_N)} \ell(\frac{1}{k}(\mathbf{z}_{k-1} + \mathbf{q}))$
    # Step II: add neuron to the current subnetwork
    $\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{q}_k$
    # Step III: uniform average of neuron outputs
    $\mathbf{u}_k = \frac{1}{k}\mathbf{z}_k$
end
Stopping Criterion: $\ell(\mathbf{u}_k) \leq \epsilon$
```

Centralized Setting. We first consider a centralized setting in which greedy forward selection is performed on a single compute node. We assume the existence of a dense, two-layer network of width N, from which the pruned model is constructed. For now, *no assumption is made regarding the amount of pre-training for the dense model.* Given a subset of neuron indices from the dense model $\mathcal{S} \subseteq [N]$, we denote the output of the pruned subnetwork that only contains this subset of neurons as follows:

$$f_{\mathcal{S}}(\mathbf{x}, \mathbf{\Theta}) = \tfrac{1}{|S|} \sum_{i \in \mathcal{S}} \sigma(\mathbf{x}, \boldsymbol{\theta}_i). \tag{8}$$

Beginning from an empty subnetwork (i.e., $\mathcal{S} = \emptyset$), we aim to discover a subset of neurons $\mathcal{S}^\star = \arg\min_{\mathcal{S} \subseteq [N]} \mathcal{L}[f_{\mathcal{S}}]$ such that $|\mathcal{S}^\star| \ll N$. In other words, we are trying to solve the following objective.

$$\min_{\mathbf{z} \in \mathcal{M}_N} \ell(\mathbf{z}) \tag{9}$$

We want to find a convex combination of neuron activations that minimizes an empirical regression loss over dataset D. Instead of discovering an exact solution to this difficult combinatorial optimization problem, greedy forward selection [251] is used to find an approximate solution. At each iteration k, we select the neuron that yields the largest decrease in loss:

$$\mathcal{S}_{k+1} = \mathcal{S}_k \cup i^\star, \;\; i^\star = \operatorname*{arg\,min}_{i \in [N]} \mathcal{L}[f_{\mathcal{S}_k \cup i}]. \tag{10}$$

Using constructions from Section 3.2, we can write the update rule for greedy forward selection as:

$$\text{(Select new neuron):} \quad \mathbf{q}_k = \operatorname*{arg\,min}_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_n)} \ell\left(\tfrac{1}{k} \cdot (\mathbf{z}_{k-1} + \mathbf{q})\right) \tag{11}$$

$$\text{(Add neuron to subnetwork):} \quad \mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{q}_k \tag{12}$$

$$\text{(Uniform average of neuron outputs):} \quad \mathbf{u}_k = \frac{1}{k} \cdot \mathbf{z}_k. \tag{13}$$

In words, (11)-(13) include the output of a new neuron, given by $\mathbf{q}_k$, within the current subnetwork at each pruning iteration based on a greedy minimization of the training loss $\ell(\cdot)$. Then, the output of the pruned subnetwork over the entire dataset at the k-th iteration, given by $\mathbf{u}_k \in \mathcal{M}_N$, is computed by taking a uniform average over the activation vectors of the k active neurons in $\mathbf{z}_k$. This pruning procedure, outlined in Algorithm 2, is the same as that of [251], but more explicitly matches the procedure in (10) by adopting a fixed, uniform average over selected neurons at each iteration. Within our analysis, rewriting the update rule as in (11)-(13) makes the comparison of subnetwork and dense network loss intuitive, *as the output of both dense and pruned networks is formulated as a uniform average over neuron activations*.

Notably, the greedy forward selection procedure in (11)-(13) can select the same neuron multiple times during successive pruning iterations. Such selection with replacement could be interpretted as a form of training during pruning—multiple selections of the same neuron is equivalent to modifying the neuron's output layer weight b_i in (4). Nonetheless, we highlight that such "training" does not violate the core purpose and utility of pruning: *we still obtain a smaller subnetwork with performance comparable to the dense network from which it was derived.*

Distributed Setting. Next, we consider a distributed setting in which the greedy forward selection process is distributed across multiple compute nodes. Recall that the set of neurons considered by greedy forward selection is given by $\texttt{Vert}(\mathcal{M}_N) = \{\mathbf{\Phi}_i : i \in [N]\}$. In the distributed setting, we assume that the weights associated with each neuron are uniformly and disjointly partitioned across compute sites. More formally, for $j \in [V]$, we define $\mathcal{A}^{(j)}$ as the indices of neurons on v_j and $\mathbf{\Theta}^{(j)} = \{\theta_i : i \in \mathcal{A}^{(j)}\}$ as the neuron weights contained on v_j. Going further, we consider $\{\mathbf{\Phi}_i : i \in \mathcal{A}^{(j)}\}$ and denote the convex hull over this subset of neuron activations as $\mathcal{M}_N^{(j)}$. We assume that $\mathcal{A}^{(j)} \bigcap \mathcal{A}^{(k)} = \varnothing$ for $j \neq k$ and that $\bigcup_{j=1}^{V} \mathcal{A}^{(j)} = [N]$.

Algorithm 3 aims to solve the objective given by (9) but in the distributed setting. We maintain a global set of active neurons throughout the pruning process that is shared across compute nodes, denoted as $\mathcal{S}_k$ at pruning iteration k. At each pruning iteration k, we perform a local search over the neurons on each $v_j \in \mathcal{V}$, then aggregate the results of these local searches and add a single neuron (i.e., the best option found by any local search) into the global set. Intuitively, Algorithm 3 adopts the same greedy forward selection process from Algorithm 2 but runs it in parallel across compute nodes.

Algorithm 3: Distributed Greedy Forward Selection for Two-Layer Networks

```
$\mathbf{z}_0^{(j)} = \mathbf{0},\ \forall j \in [V]$
for $k = 1, 2, \dots$ do
    # Step I: compute a local estimate of the next iterate
    for $v_i \in \mathcal{V}$ do
        $\mathbf{q}_k^{(i)} = \underset{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N^{(i)})}{\arg\min} \ \ell(\frac{1}{k}(\mathbf{z}_{k-1} + \mathbf{q}))$
        $\mathbf{z}_k^{(i)} = \mathbf{z}_{k-1} + \mathbf{q}_k^{(i)}$
        Broadcast: $L^{(i)} = \ell(\mathbf{z}_k^{(i)})$
    end
    # Step II: determine and broadcast the best local iterate
    for $v_i \in \mathcal{V}$ do
        $i_k = \underset{i \in [V]}{\arg\min}\, L^{(i)}$
        if $i_k == i$ then
            Broadcast: $\mathbf{z}_k = \mathbf{z}_k^{(i)}$
        end
    end
    # Step III: update the current, global iterate
    for $v_i \in \mathcal{V}$ do
        $\mathbf{u}_k = \frac{1}{k}\mathbf{z}_k$
    end
end
Stopping Criterion: $\ell(\mathbf{u}_k) \leq \epsilon$
```

3.4 How much pre-training do we really need?

As previously stated, no existing theoretical analysis has quantified the impact of pre-training on the performance of a pruned subnetwork. Here, we solve this problem by extending existing analysis for pruning via greedy forward selection to determine the relationship between SGD pre-training and subnetwork training loss. Full proofs are deferred to Appendix B.1, but we provide a sketch of the analysis within this section. We begin with all assumptions that are necessary for our analysis.

Assumption 1. *For some constant $\delta \in \mathbb{R}$, we assume the first and second derivatives of the network's activation function are bounded, such that $|\sigma'_+(\cdot)| \leq \delta$ and $|\sigma''_+(\cdot)| \leq \delta$ for σ_+ defined in* (4).

In comparison to [251], Assumption 1 lessens restrictions on the activation function[6] and removes boundedness assumptions on data and labels. Under these assumptions, we provide a bound for the loss of a subnetwork obtained after k iterations of greedy forward selection.

[6]We only require the first and second-order derivatives of the activation function to be bounded, whereas [251] imposes a Lipschitz bound and a bound on the maximum output value of the activation function.

Lemma 1. *The following expression for the objective loss is true for iterates derived after k iterations of the update rule in* (11)-(13) *with a two-layer network of width N:*

$$\ell(\mathbf{u}_k) \leq \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{k}\mathcal{L}_N \tag{14}$$

Here, $\mathcal{L}_N$ represents the loss achieved by the dense network.

This result is similar in flavor to [251]; yet, here it is derived under milder assumptions and modified to consider the loss of the dense network. Assuming $\mathcal{L}_N$ is sufficiently small, Lemma 1 tells us that the pre-defined stopping criterion ($\ell(\mathbf{u}_k) \leq \epsilon$) will be reached in $k \geq \mathcal{O}\left(\frac{1}{\epsilon}\right)$ iterations of Algorithm 2. Additionally, because Algorithm 2 selects a single neuron during each iteration, the resulting subnetwork satisfies the sparsity constraint $|\mathcal{S}_k| = \mathcal{O}\left(\frac{1}{\epsilon}\right)$.

From here, we follow the setup described in Section 3.3 to generalize Lemma 1 to the distributed case under identical assumptions.

Lemma 2. *Algorithm 3 achieves a convergence rate that is identical to that of Lemma 1 when applied to a two-layer network of width N. The algorithm terminates after $k = \mathcal{O}\left(\frac{1}{\epsilon}\right)$ iterations and $\mathcal{O}\left((Bd + VB)/\epsilon\right)$ total communication, where B is an upper bound on the total cost of broadcasting a real number to all nodes in the network G.*

This result generalizes Lemma 1 to the distributed variant of greedy forward selection. The communication overhead of Algorithm 3 is proportional to the input dimension of the neural network d and number of compute nodes V. With this in mind, distributed greedy forward selection enables the pruning process to be parallelized across multiple compute nodes with minimal communication overhead (i.e., no dependence upon the hidden dimension N) while achieving the same per-iteration convergence rate. Given that both centralized and distributed variants of greedy forward selection achieve an identical convergence rate with respect to the number of pruning iterations, all theoretical results that follow apply to both algorithms.

Because assumptions in Lemmas 1 and 2 match those of [191], we can generalize the convergence result to the stochastic case (i.e., with respect to randomness over SGD iterations) and consider the number of pre-training iterations performed on the dense network.

Theorem 1. *Assume Assumption 1 holds and that a two-layer network of width N was pre-trained for t iterations with SGD over a dataset D of size m. Additionally, assume that $Nd > m^2$ and $m > d$, where d represents the input dimension of data in D.*[7] *Then, a subnetwork obtained from this dense network via k iterations of greedy forward selection satisfies the following:*

$$\mathbb{E}[\ell(\mathbf{u}_k)] \leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \mathcal{L}_0,$$

where $\mathcal{L}_0$ is the loss of the dense network at initialization, c and ζ are positive constants, and all expectations are with respect to randomness over SGD iterations on the dense network.

[7]This overparameterization assumption is mild in comparison to previous work [8, 60]. Only recently was an improved, subquadratic requirement derived [218].

Trivially, the loss in Theorem 1 only decreases during successive pruning iterations if the rightmost $\mathcal{L}_0$ term does not dominate the expression (i.e., all other terms decay as $\mathcal{O}(\frac{1}{k})$). If this term does not decay with increasing k, the upper bound on subnetwork training loss deteriorates, thus eliminating any guarantees on subnetwork performance. Interestingly, we discover that the tightness of the upper bound in Theorem 1 depends on the number of dense network SGD pre-training iterations as follows.[8]

Theorem 2. *Adopting identical assumptions and notation as Theorem 1, assume the dense network is pruned via greedy forward selection for k iterations. The resulting subnetwork is guaranteed to achieve a training loss $\propto \mathcal{O}(\frac{1}{k})$ if t—the number of SGD pre-training iterations on the dense network—satisfies the following condition:*

$$t \gtrapprox \mathcal{O}\left(\frac{-\log(k)}{\log\left(1 - c\frac{d}{m^2}\right)}\right), \quad \textit{where } c \textit{ is a positive constant.} \tag{15}$$

Otherwise, the loss of the pruned network is not guaranteed to improve over successive iterations of greedy forward selection.

Discussion.[9] Our $\mathcal{O}(\frac{1}{k})$ rate in Lemmas 1 and 2 matches that of previous work under mild assumptions, and explicitly expresses the loss of the subnetwork with respect to the loss of the dense network. Assumption 1 is chosen to align with the analysis of [191]. This combination enables the training loss of the pruned subnetwork to be expressed with respect to the number of pre-training iterations on the dense network as shown in Theorem 1.

Theorem 2 states that *a threshold exists in the number of SGD pre-training iterations of a dense network, beyond which subnetworks derived with greedy selection are guaranteed to achieve good training loss*. Denoting this threshold as $t^\star$, in the case that $m^2 \gg d$, $\lim_{m\to\infty} t^\star = \infty$, implying that larger datasets require more pre-training for high-performing subnetworks to be discovered. When $m^2 \approx d$, $\lim_{m\to cd} t^\star = 0$; i.e., minimal pre-training is required for subnetworks to perform well on small datasets. Interestingly, $t^\star$ scales logarithmically with the dataset size, implying that the amount of required pre-training will plateau as the underlying dataset becomes increasingly large.

Our finding provides theoretical insight regarding $i)$ how much pre-training is sufficient to discover a subnetwork that performs well and $ii)$ the difficulty of discovering high-performing subnetworks in large-scale experiments (i.e., large datasets require more pre-training). Because several empirical heuristics for discovering high-performing subnetworks without full pre-training have already been proposed [253, 254], we choose to not focus on deriving novel empirical methods. Nonetheless, we do propose a distributed variant of the greedy forward selection algorithm to accelerate the pruning process (see Section 3.6.1) for which all theoretical results hold. Our results specifically focus upon training accuracy, while generalization performance could be a more realistic assessment of subnetwork quality. It is possible that our analysis can be

[8] We also derive a similar result using gradient descent (GD) instead of SGD; see Appendix B.1.

[9] A faster rate can be achieved with greedy forward selection given Assumption 1 if $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$; see Appendix B.1.4. This result still holds under our milder assumptions, but the proof is similar to [251]. It is true that such a result indicates that subnetworks perform well without pre-training when $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$, which seems to lessen the significance of Theorem 2. However, we analyze this assumption practically within Appendix B.2 and show that, even for extremely small scale experiments, the assumption $i)$ never holds at initialization and $ii)$ tends to require more training for larger datasets, which aligns with findings in Theorem 2.

combined with theoretical results for neural network generalization performance [10, 150], but we leave such analysis as future work.

3.5 Related Work

Neural Network Pruning. Many variants have been proposed for both structured [93, 148, 162, 251] and unstructured [65, 66, 71, 94] pruning. Generally, structured pruning, which prunes entire channels or neurons of a network instead of individual weights, is considered more practical, as it can achieve speedups without leveraging specialized libraries for sparse computation. Existing pruning criterion include the norm of weights [148, 162], feature reconstruction error [110, 170, 252, 257], or even gradient-based sensitivity measures [17, 232, 273]. While most pruning methodologies perform backward elimination of neurons within the network [71, 72, 162, 163, 257], some recent research has focused on forward selection structured pruning strategies [251, 252, 273]. We adopt greedy forward selection within this work, as it has been previously shown to yield superior performance in comparison to greedy backward elimination.

Frank-Wolfe Algorithm. Our analysis resembles that of the Frank-Wolfe algorithm [70, 129], a widely-used technique for constrained, convex optimization. Recent work has shown that training deep networks with Frank-Wolfe can be made feasible in certain cases despite the non-convex nature of neural network training [12, 196]. Instead of training networks from scratch with Frank-Wolfe, however, we use a Frank-Wolfe-style approach to greedily select neurons from a pre-trained model. Such a formulation casts structured pruning as convex optimization over a marginal polytope, which can be analyzed similarly to Frank-Wolfe [251, 252] and loosely approximates networks trained with standard, gradient-based techniques [251]. Several distributed variants of the Frank-Wolfe algorithm have been analyzed theoretically [117, 230, 246], though our analysis most closely resembles that of [19]. Alternative methods of analysis for greedy selection algorithms could also be constructed with the use of sub-modular optimization techniques [185].

Training Analysis. Much work has been done to analyze the convergence properties of neural networks trained with gradient-based techniques [32, 96, 127, 267]. Such convergence rates were originally explored for wide, two-layer neural networks using mean-field analysis techniques [176, 219]. Similar techniques were later used to extend such analysis to deeper models [168, 247]. Generally, recent work on neural network training analysis has led to novel analysis techniques [96, 127], extensions to alternate optimization methodologies [128, 191], and even generalizations to different architectural components [85, 152, 267]. By adopting and extending such analysis, we aim to bridge the gap between the theoretical understanding of neural network training and LTH.

3.6 Experiments

In this section, we empirically validate our theoretical results from Section 3.4, which predict that larger datasets require more pre-training for subnetworks obtained via greedy forward selection to perform well. Pruning via greedy forward selection has already been empirically analyzed in previous work. Therefore, *we differentiate our experiments by providing an in-depth analysis of the scaling properties of greedy forward selection with respect to the size and complexity of the underlying dataset.* In particular, we produce synthetic

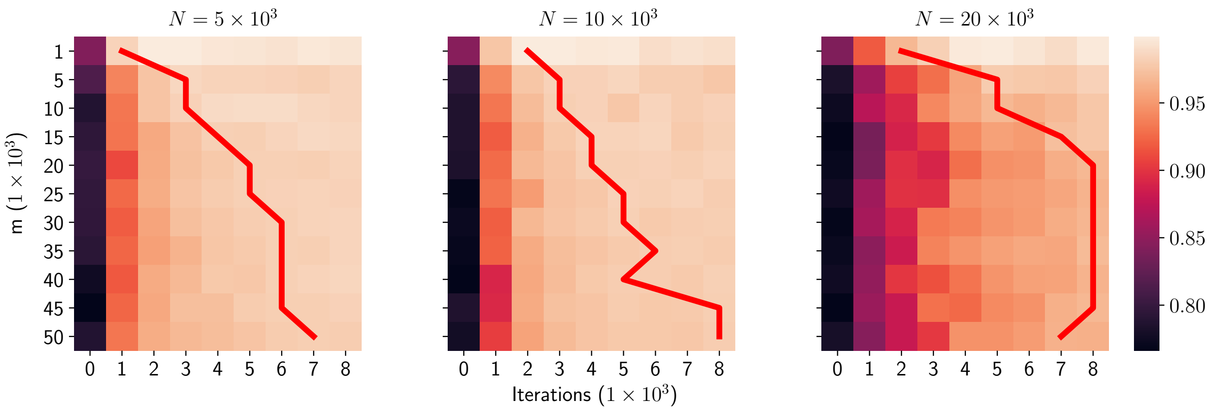

Figure 4: Pruned, two-layer models on MNIST. Sub-plots depict different dense network sizes, while the x and y axis depict the number of pre-training iterations and the sub-dataset size, respectively. Models are pruned to 200 hidden neurons every 1K iterations to measure subnetwork performance. Color represents training accuracy, and the red line depicts the point at which subnetworks surpass the performance of the best pruned model on the full dataset for different sub-dataset sizes.

"sub-datasets" of different sizes and measure the amount of pre-training required to discover a high-performing subnetwork on each.

We perform analysis with two-layer networks on MNIST [52] and with deep CNNs (i.e., ResNet34 [106] and MobileNetV2 [207]) on CIFAR10 and ImageNet [51, 144]. We find that $i)$ *high-performing subnetworks can be consistently discovered without incurring the full pre-training cost*, and $ii)$ *the amount of pre-training required for a subnetwork to perform well increases with the size and complexity of the underlying dataset in practice*. All experiments are run on an internal cluster with two Nvidia RTX 3090 GPUs using the public implementation of greedy forward selection [250]. On large-scale experiments (i.e., those performed on ImageNet), we improve pruning efficiency by parallelizing the greedy forward selection process across two GPUs according to Algorithm 3.

3.6.1 Distributed Greedy Forward Selection

As outlined in Section 3.4, the centralized and distributed variants of greedy forward selection achieve identical convergence rates with respect to the number of pruning iterations. Despite its impressive empirical results, one of the major drawbacks of greedy forward selection is that it is slow and computationally expensive compared to heuristic techniques. Distributed greedy forward selection mitigates this problem by parallelizing the pruning process across multiple compute nodes with minimal communication overhead.

To practically examine the acceleration provided by distributed greedy forward selection, we prune a ResNet34 architecture [106] on the ImageNet dataset and measure the pruning time for each layer with different greedy forward selection variants. In particular, we select four blocks from the ResNet34 architecture with different spatial and channel dimensions. The time taken to prune each of these blocks is shown in Figure 5.

Distributed greedy forward selection (using either two or four GPUs) significantly accelerates the pruning process for nearly all blocks within the ResNet. Notably, no speedup is observed for the second block because

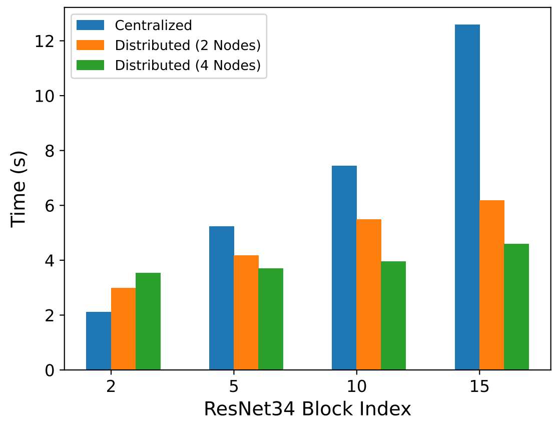

Figure 5: Pruning time for centralized and distributed greedy forward selection applied to different blocks of a ResNet34 architecture on ImageNet.

earlier ResNet layers have fewer channels to be considered by greedy forward selection. As the channel dimension increases in later layers, distributed greedy forward selection yields a significant speedup in the pruning process. Given that the convergence guarantees of distributed greedy forward selection are identical to those of the centralized variant, we adopt the distributed algorithm to improve efficiency in the majority of our large-scale pruning experiments.

3.6.2 Two-Layer Networks

We perform structured pruning experiments with two-layer networks on MNIST [52] by pruning hidden neurons via greedy forward selection. To match the single output neuron setup described in Section 3.2, we binarize MNIST labels by considering all labels less than five as zero and vice versa. Our model architecture matches the description in Section 3.2 with a few minor differences. Namely, we adopt a ReLU hidden activation and apply a sigmoid output transformation to enable training with binary cross entropy loss. Experiments are conducted with several different hidden dimensions (i.e., $N \in \{5\text{K}, 10\text{K}, 20\text{K}\}$). Hyperparameters are tuned using a hold out validation set; see Appendix B.3.1 for more details.

To study how dataset size impacts subnetwork performance, we construct sub-datasets of sizes 1K to 50K (i.e., in increments of 5K) from the original MNIST dataset by uniformly sampling examples from the 10 original classes. The two-layer network is pre-trained for 8K iterations in total and pruned every 1K iterations to a size of 200 hidden nodes; see Appendix B.3.1 for a precise, algorithmic description of the two-layer network pruning process. After pruning, the accuracy of the pruned model over the entire training dataset is recorded (i.e., no fine-tuning is performed), allowing the impact of dataset size and pre-training length on subnetwork performance to be observed. See Figure 4 for these results, which are averaged across three trials.

Discussion. The performance of pruned subnetworks in Figure 4 matches the theoretical analysis provided in Section 3.4 for all different sizes of two-layer networks. Namely, *as the dataset size increases, so does the amount of pre-training required to produce a high-performing subnetwork.* To see this, one can track the trajectory of the red line, which traces the point at which the accuracy of the best performing subnetwork

Model	Dataset Size	Pruned Accuracy				Dense Accuracy
		20K It.	40K It.	60K It.	80K It.	
MobileNetV2	10K	82.32	86.18	86.11	86.09	83.13
	30K	80.19	87.79	88.38	88.67	87.62
	50K	86.71	88.33	91.79	91.77	91.44
ResNet34	10K	75.29	85.47	85.56	85.01	85.23
	30K	84.06	91.59	92.31	92.15	92.14
	50K	89.79	91.34	94.28	94.23	94.18

Table 5: CIFAR10 test accuracy for subnetworks derived from dense networks with varying pre-training amounts (i.e., number of training iterations listed in top row) and sub-dataset sizes.

for the full dataset is surpassed at each sub-dataset size. This trajectory clearly illustrates that pre-training requirements for high-performing subnetworks increase with the size of the dataset. Furthermore, this increase in the amount of required pre-training is seemingly logarithmic, as the trajectory typically plateaus at larger dataset sizes.

Interestingly, despite the use of a small-scale dataset, high-performing subnetworks are never discovered at initialization, revealing that some minimal amount of pre-training is often required to obtain a good subnetwork via greedy forward selection. Previous work claims that high-performing subnetworks may exist at initialization in theory. In contrast, our empirical analysis shows that this is not the case even in simple experimental settings.

3.6.3 Deep Networks

We perform structured pruning experiments (i.e., channel-based pruning) using ResNet34 [106] and MobileNetV2 [207] architectures on CIFAR10 and ImageNet [51, 144]. We adopt the same generalization of greedy forward selection to pruning deep networks as described in [251] and use ϵ to denote our stopping criterion; see Appendix B.3.2 for a complete algorithmic description. We follow the three-stage methodology—pre-training, pruning, and fine-tuning—and modify both the size of the underlying dataset and the amount of pre-training prior to pruning to examine their impact on subnetwork performance. Standard data augmentation and splits are adopted for both datasets.

CIFAR10. Three CIFAR10 sub-datasets of size 10K, 30K, and 50K (i.e., full dataset) are created using uniform sampling across classes. Pre-training is conducted for 80K iterations using SGD with momentum and a cosine learning rate decay schedule starting at 0.1. We use a batch size of 128 and weight decay of $5 \cdot 10^{-4}$.[10] The dense model is independently pruned every 20K iterations, and subnetworks are fine-tuned for 2500 iterations with an intial learning rate of 0.01 prior to being evaluated. We adopt $\epsilon = 0.02$ and $\epsilon = 0.05$ for MobileNet-V2 and ResNet34, respecitvely, yielding subnetworks with a 40% decrease in FLOPS and 20% decrease in model parameters in comparison to the dense model.[11]

[10]Our pre-training settings are adopted from a popular repository for the CIFAR10 dataset [160].

[11]These settings are derived using a grid search over values of ϵ and the learning rate with performance measured over a hold-out validation set; see Appendix B.3.2.

Model	**FLOP (Param)**	**Pruned Accuracy**			**Dense Accuracy**
	Ratio	**50 Epoch**	**100 Epoch**	**150 Epoch**	
MobileNetV2	60% (80%)	70.05	71.14	71.53	71.70
	40% (65%)	69.23	70.36	71.10	
ResNet34	60% (80%)	71.68	72.56	72.65	73.20
	40% (65%)	69.87	71.44	71.33	

Table 6: Test accuracy on ImageNet of subnetworks with different FLOP levels derived from dense models with varying amounts of pre-training (i.e., training epochs listed in top row). We report the FLOP/parameter ratio after pruning with respect to the FLOPS/parameters of the dense model.

The results of these experiments are presented in Table 5. The amount of training required to discover a high-performing subnetwork consistently increases with the size of the dataset. For example, with MobileNetV2, a winning ticket is discovered on the 10K and 30K sub-datasets in only 40K iterations, while for the 50K sub-dataset a winning ticket is not discovered until 60K iterations of pre-training have been completed. Furthermore, subnetwork performance often surpasses the performance of the fully-trained dense network *without completing the full pre-training procedure*.

ImageNet. We perform experiments on the ILSVRC2012, 1000-class dataset [51] to determine how pre-training requirements change for subnetworks pruned to different FLOP levels.[12] We adopt the same experimental and hyperparameter settings as [251]. Models are pre-trained for 150 epochs using SGD with momentum and cosine learning rate decay with an initial value of 0.1. We use a batch size of 128 and weight decay of $5 \cdot 10^{-4}$. The dense network is independently pruned every 50 epochs, and the subnetwork is fine-tuned for 80 epochs using a cosine learning rates schedule with an initial value of 0.01 before being evaluated. We first prune models with $\epsilon = 0.02$ and $\epsilon = 0.05$ for MobileNetV2 and ResNet34, respectively, yielding subnetworks with a 40% reduction in FLOPS and 20% reduction in parameters in comparison to the dense model. Pruning is also performed with a larger ϵ value (i.e., $\epsilon = 0.05$ and $\epsilon = 0.08$ for MobileNetV2 and ResNet34, respectively) to yield subnetworks with a 60% reduction in FLOPS and 35% reduction in model parameters in comparison to the dense model.

The results are reported in Table 6. Although the dense network is pre-trained for 150 epochs, subnetwork test accuracy reaches a plateau after only 100 epochs of pre-training in all cases. Furthermore, subnetworks with only 50 epochs of pre-training still perform well in many cases. E.g., the 60% FLOPS ResNet34 subnetwork with 50 epochs of pre-training achieves a testing accuracy within 1% of the pruned model derived from the fully pre-trained network. Thus, *high-performing subnetworks can be discovered with minimal pre-training even on large-scale datasets like ImageNet*.

Discussion. These results demonstrate that the number of dense network pre-training iterations needed to reach a plateau in subnetwork performance $i)$ consistently increases with the size of the dataset and $ii)$ is consistent across different architectures given the same dataset. Discovering a high-performing subnetwork on the ImageNet dataset takes roughly 500K pre-training iterations (i.e., 100 epochs). In comparison, discovering

[12]We do not experiment with different sub-dataset sizes on ImageNet due to limited computational resources.

a subnetwork that performs well on the MNIST and CIFAR10 datasets takes roughly 8K and 60K iterations, respectively. Thus, the amount of required pre-training iterations increases based on the size of dataset *even across significantly different scales and domains*. This indicates that dependence of pre-training requirements on dataset size may be an underlying property of discovering high-performing subnetworks no matter the experimental setting.

Per Theorem 2, the size of the dense network will not impact the number of pre-training iterations required for a subnetwork to perform well. This is observed to be true within our experiments; e.g., MobileNet and ResNet34 reach plateaus in subnetwork performance at similar points in pre-training for CIFAR10 and ImageNet in Tables 5 and 6. However, the actual loss of the subnetwork, as in Lemma 1, has a dependence on several constants that may impact subnetwork performance despite having no aymptotic impact on Theorem 2. E.g., a wider network could increase the width of the polytope $D_{\mathcal{M}_N}$ or initial loss $\ell(\mathbf{u}_1)$, leading to a looser upper bound on subnetwork loss. Thus, different sizes of dense networks, despite both reaching a plateau in subnetwork performance at the same point during pre-training, may yield subnetworks with different performance levels.

Interestingly, we observe that dense network size does impact subnetwork performance. In Figure 4, subnetwork performance varies based on dense network width, and subnetworks derived from narrower dense networks seem to achieve better performance. Similarly, in Tables 5 and 6, subnetworks derived from MobileNetV2 tend to achieve higher relative performance with respect to the dense model. Thus, subnetworks derived from smaller dense networks seem to achieve better *relative* performance in comparison to those derived from larger dense networks, suggesting that pruning via greedy forward selection may demonstrate different qualities in comparison to more traditional approaches (e.g., iterative magnitude-based pruning [163]). Despite this observation, however, the amount of pre-training epochs required for the emergence of the best-performing subnetwork is still consistent across architectures and dependent on dataset size.

3.7 Conclusion

In this work, we theoretically analyze the impact of dense network pre-training on the performance of a pruned subnetwork obtained via a centralized or distributed variant of greedy forward selection. By expressing pruned network loss with respect to the number of SGD iterations performed on its associated dense network, we discover a threshold in the number of pre-training iterations beyond which a pruned subnetwork achieves good training loss. Furthermore, we show that this threshold is logarithmically dependent upon the size of the dataset, which offers intuition into the early-bird ticket phenomenon and the difficulty of replicating pruning experiments at scale. We empirically verify our theoretical findings over several datasets (i.e., MNIST, CIFAR10, and ImageNet) with numerous network architectures (i.e., two-layer networks, MobileNetV2, and ResNet34), showing that the amount of pre-training required to discover a winning ticket is consistently dependent on the size of the underlying dataset. Several open problems remain, such as extending our analysis beyond two-layer networks, deriving generalization bounds for subnetworks pruned with greedy forward selection, or even using our theoretical results to discover new heuristic methods for identifying early-bird tickets in practice.

4 i-SpaSP: Structured Neural Pruning via Sparse Signal Recovery

We propose a novel, structured pruning algorithm for neural networks—the **i**terative, **Spa**rse **S**tructured **P**runing algorithm, dubbed as i-SpaSP. Inspired by ideas from sparse signal recovery, i-SpaSP operates by iteratively identifying a larger set of important parameter groups (e.g., filters or neurons) within a network that contribute most to the residual between pruned and dense network output, then thresholding these groups based on a smaller, pre-defined pruning ratio. For both two-layer and multi-layer network architectures with ReLU activations, we show the error induced by pruning with i-SpaSP decays polynomially, where the degree of this polynomial becomes arbitrarily large based on the sparsity of the dense network's hidden representations. In our experiments, i-SpaSP is evaluated across a variety of datasets (i.e., MNIST, ImageNet, and XNLI) and architectures (i.e., feed forward networks, ResNet34, MobileNetV2, and BERT), where it is shown to discover high-performing sub-networks and improve upon the pruning efficiency of provable baseline methodologies by several orders of magnitude. Put simply, i-SpaSP is easy to implement with automatic differentiation, achieves strong empirical results, comes with theoretical convergence guarantees, and is efficient, thus distinguishing itself as one of the few computationally efficient, practical, and provable pruning algorithms.

4.1 Introduction

Neural network pruning has garnered significant recent interest [71, 148, 163], as obtaining high-performing sub-networks from larger, dense networks enables a reduction in the computational and memory overhead of neural network applications [94, 95]. Many popular pruning techniques are based upon empirical heuristics that work well in practice [72, 110, 170]. Generally, these methodologies introduce some notion of "importance" for network parameters (or groups of parameters) and eliminate parameters with negligible importance.

The empirical success of pruning methodologies inspired the development of pruning algorithms with theoretical guarantees [16, 17, 156, 181, 200]. Among such work, greedy forward selection (GFS) [251, 252]—inspired by the Frank-Wolfe algorithm [70]—differentiated itself as a methodology that performs well in practice and provides theoretical guarantees. However, GFS is inefficient in comparison to popular pruning heuristics.

Our Proposal. We leverage greedy selection [139, 184] to develop a structured pruning algorithm that is provable, practical, and efficient. This algorithm, called **i**terative, **Spa**rse **S**tructured **P**runing (i-SpaSP), iteratively estimates the most important parameter groups[13] within each network layer, then thresholds this set of parameter groups based on a pre-defined pruning ratio—a similar procedure to the CoSAMP algorithm for sparse signal recovery [184]. Theoretically, we show for two and multi-layer networks that $i)$ the output residual between pruned and dense networks decays polynomially with respect to the size of the pruned network and $ii)$ the order of this polynomial increases as the dense network's hidden representations become more sparse.[14] In experiments, we show that i-SpaSP is capable of discovering high-performing sub-networks

[13]"Parameter groups" refers to the minimum structure used for pruning (e.g., neurons or filters).

[14]To the best of the authors' knowledge, we are the first to provide theoretical analysis showing that the quality of pruning depends upon the sparsity of representations within the dense network.

across numerous different models (i.e., two-layer networks, ResNet34, MobileNetV2, and BERT) and datasets (i.e., MNIST, ImageNet, and XNLI). i-SpaSP is simple to implement and significantly improves upon the runtime of GFS variants.

4.2 Preliminaries

Notation. Vectors and scalars are denoted with lower-case letters. Matrices and certain constants are denoted with upper-case letters. Sets are denoted with upper-case, calligraphic letters (e.g., $\mathcal{G}$) with set complements $\mathcal{G}^c$. We denote $[n] = \{0, 1, \dots, n\}$. For $x \in \mathbb{R}^N$, $\|x\|_p$ is the ℓ_p vector norm. x_s is the s largest-valued components of x, where $|x| \geq s$. $\text{supp}(x)$ returns the support of x. For index set $\mathcal{G}$, $x|_{\mathcal{G}}$ is the vector with non-zeros at the indices in $\mathcal{G}$. For $X \in \mathbb{R}^{m \times n}$, $\|X\|_F$ and $X^\top$ represent the Frobenius norm and transpose of X. The i-th row and j-th column of X are given by $X_{i,:}$ and $X_{:,j}$, respectively. $\text{rsupp}(X)$ returns the row support of X (i.e., indices of non-zero rows). For index set $\mathcal{G}$, $X_{\mathcal{G},:}$ and $X_{:,\mathcal{G}}$ represent row and column sub-matrices, respectively, that contain rows or columns with indices in $\mathcal{G}$. $\mu(X) : \mathbb{R}^{m \times n} \longrightarrow \mathbb{R}^m$ sums over columns of a matrix (i.e., $\mu(X) = \sum_i X_{:,i}$), while $\texttt{vec}(X) : \mathbb{R}^{m \times n} \longrightarrow \mathbb{R}^{mn}$ stacks columns of a matrix. $X \in \mathbb{R}^{m \times n}$ is p-row-compressible with magnitude $R \in \mathbb{R}^{\geq 0}$ if $|\mu(X)|_{(i)} \leq \frac{R}{i^{\frac{1}{p}}}, \;\; \forall i \in [m]$, where $|\cdot|_{(i)}$ denotes the i-th sorted vector component (in magnitude). Lower p values indicate a nearly row-sparse matrix and vice versa.

Network Architecture. Our analysis primarily considers two-layer, feed forward networks[15]:

$$f(X, \mathcal{W}) = W^{(1)} \cdot \sigma(W^{(0)} \cdot X) \tag{16}$$

The network's input, hidden, and output dimensions are given by d_{in}, d_{hid}, and d_{out}. $X \in \mathbb{R}^{d_{in} \times B}$ stores the full input dataset with B examples. $W^{(0)} \in \mathbb{R}^{d_{hid} \times d_{in}}$ and $W^{(1)} \in \mathbb{R}^{d_{out} \times d_{hid}}$ denote the network's weight matrices. $\sigma(\cdot)$ denotes the ReLU activation function and $H = \sigma(W^0 \cdot X)$ stores the network hidden representations across the dataset. We also extend our analysis to multi-layer networks with similar structure; see the Appendix for more details.

4.3 Related Work

Pruning. Neural network pruning strategies can be roughly separated into structured [93, 148, 162, 251, 252] and unstructured [65, 66, 71, 94] variants. Structured pruning, as considered in this work, prunes parameter groups instead of individual weights, allowing speedups to be achieved without sparse computation [148]. Empirical heuristics for structured pruning include removing parameter groups with low ℓ_1 norm [148, 162], measuring the gradient-based sensitivity of parameter groups [17, 232, 273], preserving network output [110, 170, 257], and more [44, 123, 179, 223]. Pruning typically follows a three-step process of pre-training, pruning, and fine-tuning [148, 163], where pre-training is typically the most expensive component [41, 254].

Provable Pruning. Empirical pruning research inspired the development of theoretical foundations for network pruning, including sensitivity-based analysis [17, 156], coreset methodologies [16, 181], random

[15]Though (16) has no bias, a bias term could be implicitly added as an extra element within the input and weight matrices.

network pruning analysis [173, 190, 195, 200], and generalization analysis [265]. GFS—analyzed for two-layer [251] and multi-layer [252] networks—was one of the first pruning methodologies to provide both strong empirical performance and theoretical guarantees.

Greedy Selection. Greedy selection efficiently discovers approximate solutions to combinatorial optimization problems [70]. Many algorithms and frameworks for greedy selection exist; e.g., Frank-Wolfe [70], sub-modular optimization [185], CoSAMP [184], and iterative hard thresholding [139]. Frank-Wolfe has been used within GFS and to train deep neural networks [12, 196], thus forming a connection between greedy selection and deep learning. Furthering this connection, we leverage CoSAMP [184] to formulate our proposed methodology.

4.4 Methodology

i-SpaSP is formulated for two-layer networks in Algorithm 4, where the pruned model size s and total iterations T are fixed. $\mathcal{S}$ stores active neurons in the pruned model, which is refined over iterations.

Algorithm 4: i-SpaSP for Two-Layer Networks

```
Parameters: $T$, $s$; $\mathcal{S} := \emptyset$; $t := 0$
# compute hidden representation
$H = \sigma(W^{(0)} \cdot X)$
$h = \mu(H)$
# compute dense network output
$U = W^{(1)} \cdot H$
$V = U$
while $t < T$ do
    $t = t + 1$
    # Step I: Estimating Importance
    $Y = (W^{(1)})^\top \cdot V$
    $y = \mu(Y)$
    $\Omega = \mathrm{supp}(y_{2s})$
    # Step II: Merging and Pruning
    $\Omega^\star = \Omega \cup \mathcal{S}$
    $b = h|_{\Omega^\star}$
    $\mathcal{S} = \mathrm{supp}(b_s)$
    # Step III: Computing New Residual
    $V = U - W^{(1)}_{:,\mathcal{S}} \cdot H_{\mathcal{S},:}$
end
# return pruned model with neurons in $\mathcal{S}$
return $\{W^{(0)}_{\mathcal{S},:}, W^{(1)}_{:,\mathcal{S}}\}$
```

Why does this work? Each iteration of i-SpaSP follows a three-step procedure in Algorithm 4:

Step I: Compute neuron "importance" Y given the current residual matrix V.

Step II: Identify s neurons within the combined set of important and active neurons (i.e., $\Omega \cup \mathcal{S}$) with the largest-valued hidden representations.

Step III: Update V with respect to the new pruned model estimate.

We now provide intuition regarding the purpose of each individual step within i-SpaSP.

Estimating Importance. Y_{ij} is the importance of hidden neuron i with respect to dataset example j. We can characterize the discrepancy between pruned and dense network output U' and U as:

$$\mathcal{L}(U, U') = \frac{1}{2}\|W^{(1)} \cdot H - U'\|_F^2. \tag{17}$$

Considering U' fixed, $\nabla_H \mathcal{L}(U, U') = (W^{(1)})^\top \cdot V$. As such, if Y_{ij} is a large, positive (negative) value, decreasing (increasing) H_{ij} will decrease the value of $\mathcal{L}$ locally. Then, because H is non-negative and cannot be modified via pruning, one can realize that the best methodology of minimizing (17) is including neurons with large, positive importance values within $\mathcal{S}$, as in Algorithm 4.

Merging and Pruning. The $2s$ most-important neurons—based on $\mu(Y)$ components—are selected and merged with $\mathcal{S}$, allowing a larger set of neurons (i.e., more than s) to be explored. From here, s neurons with the largest-valued components in $\mu(H)$ are sub-selected from this combined set to form the next pruned model estimate. Because hidden representation values are not considered in importance estimation, performing this two-step merging and pruning process ensures neurons within $\mathcal{S}$ have both large hidden activation and importance values, which together indicate a meaningful impact on network output.

Computing the New Residual. The next pruned model estimate is used to re-compute V, which can be intuitively seen as updating U' in (17). As such, importance in Algorithm 4 is based on the current pruned and dense network residual and (17) is minimized over successive iterations.

4.4.1 Implementation

```
$H$.requires_grad := True
with torch.no_grad():
    prune_out := prune_layer($H_{\mathcal{S},:}$)
dense_out := dense_layer($H$)
obj := sum $\left(\frac{1}{2}(\text{dense\_out} - \text{prune\_out})^2\right)$
obj.backward()
importance := sum($H$.grad, dim = 0)
return importance
```

Figure 6: i-SpaSP importance computation via automatic differentiation.

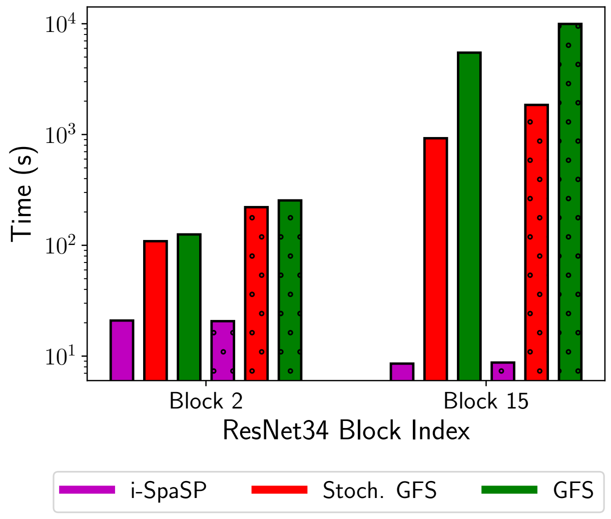

Figure 7: Runtime of pruning ResNet34 blocks with i-SpaSP and GFS variants to 20% (i.e., plain) or 40% (i.e., dotted) of filters.

Automatic Differentiation. Because $Y = \nabla_H \mathcal{L}(U, U')$, importance within i-SpaSP can be computed efficiently using automatic differentiation [1, 193]; see Figure 6 for a PyTorch-style example. Automatic differentiation simplifies importance estimation and allows it to be run on a GPU, making the implementation efficient and parallelizable. Because the remainder of the pruning process only leverages basic sorting and set operations, the overall implementation of i-SpaSP is both simple and efficient.

Other Architectures. The code in Figure 6 can be easily generalized to more complex network modules (i.e., beyond feed-forward layers) using automatic differentiation. Notably, convolutional filters or attention heads can be pruned using the importance estimation from Figure 6 if $\texttt{sum}(\cdot)$ is performed over both batch and spatial dimensions. Furthermore, i-SpaSP can be used to prune multi-layer networks by greedily pruning each layer of the network from beginning to end.

Large-Scale Datasets. Algorithm 4 assumes the entire dataset is stored within X. For large-scale experiments, such an approach is not tractable. As such, we redefine X within experiments to contain a subset or mini-batch of data from the full dataset, allowing the pruning process to be performing in an approximate—but computationally tractable—manner. To improve the quality of this approximation, a new mini-batch is sampled during each i-SpaSP iteration, but the size of such mini-batches becomes a hyperparameter of the pruning process.[16]

4.4.2 Computational Complexity and Runtime Comparisons

Denote the complexity of matrix-matrix multiplication as ξ. The complexity of pruning a network layer with T iterations of i-SpaSP is $\mathcal{O}(T\xi + Td_{hid}\log(d_{hid}))$. GFS has a complexity of $\mathcal{O}(s\xi d_{hid})$, as it adds a single neuron to the (initially empty) network layer at each iteration by exhaustively searching for the neuron that

[16]Both GFS [251] and multi-layer GFS [252] adopt a similar mini-batch approach during pruning.

minimizes training loss. Though later GFS variants achieve complexity of $\mathcal{O}(s\xi)$ [252], i-SpaSP is more efficient in practice because the forward pass (i.e. $\mathcal{O}(\xi)$) dominates the pruning procedure and $T \ll s$ (e.g., $T = 20$ in Section 4.6).

As a practical runtime comparison, we adopt a ResNet34 model [106] and measure wall-clock pruning time[17] with i-SpaSP and GFS. We use the public implementation of GFS [250] and test both stochastic and vanilla variants[18]. i-SpaSP uses settings from Section 4.6, and selected ResNet blocks are pruned to ratios of 20% or 40% of original filters; see Figure 7. i-SpaSP significantly improves upon the runtime of GFS variants; e.g., i-SpaSP prunes Block 15 in roughly 10 seconds, while GFS takes over 1000 seconds in the best case. Furthermore, unlike GFS, the runtime of i-SpaSP is not sensitive to the size of the pruned network (i.e., wall-clock time is similar for ratios of 20% and 40%), though i-SpaSP does prune later network layers faster than earlier layers.

4.5 Theoretical Results

Proofs are deferred to the Appendix. The dense network is pruned from d_{hid} to s neurons via i-SpaSP. We assume that $W^{(1)}$ satisfies the restricted isometry property (RIP) [28]:

Assumption 2 (Restricted Isometry Property (RIP) [28]). *Denote the r-th restricted isometry constant as δ_r and assume $\delta_r \leq 0.1$ for $r = 4s$.*[19] *Then, $W^{(1)}$ satisfies the RIP with constant δ_r if $(1-\delta_r)\|x\|_2^2 \leq \|W^{(1)} \cdot x\|_2^2 \leq (1+\delta_r)\|x\|_2^2$ for all $\|x\|_0 \leq r$.*

No assumption is made upon $W^{(0)}$. We hypothesize that Assumption 2 is mild due to properties like semi or quarter-circle laws that bound the eigenvalues symmetric, random matrices within some range [63], but we leave the formal verification of this assumption as future work. We define $H = \sigma(W^{(0)} \cdot X) \in \mathbb{R}^{d_{hid} \times B}$, which can be theoretically reformulated as $H = Z + E$ for s-row-sparse Z and arbitrary E. From here, we can show the following about the residual between pruned and dense network hidden representations after t iterations of Algorithm 4.

Lemma 3. *If Assumption 2 holds, the pruned approximation to H after t iterations of Algorithm 4, $H_{\mathcal{S}_t,:}$, is s-row-sparse and satisfies the following inequality:*

$$\|\mu(H - H_{\mathcal{S}_t,:})\|_2 \leq (0.444)^t \|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(E)\|_1$$

Going further, Lemma 3 can be used to bound the residual between pruned and dense network output.

Theorem 3. *Let $U = W^{(1)} \cdot H$ and $U' = W^{(1)}_{:,\mathcal{S}_t} \cdot H_{\mathcal{S}_t,:}$ denote pruned and dense network output, respectively. $V_t = U - U'$ stores the residual between pruned and dense network output over the entire dataset. If*

[17]We prune the 2nd (64 channels) and 15th (512 channels) convolutional blocks. We choose blocks in different network regions to view the impact of channel and spatial dimension on pruning efficiency.

[18]Vanilla GFS exhaustively searches neurons within each iteration, while the stochastic variant randomly selects 50 neurons to search per iteration.

[19]This is a numerical assumption adopted from [184], which holds for Gaussian matrices of size $\mathbb{R}^{m \times n}$ when $m \geq \mathcal{O}\left(r \log(\frac{n}{r})\right)$.

Assumption 2 holds, we have the following at iteration t of Algorithm 4:

$$\|V_t\|_F \leq \|W^{(1)}\|_F \cdot \left((0.444)^t \|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(E)\|_1 \right)$$

Because $\|\mu(H)\|_2$ decays linearly in Theorem 3, $\|\mu(E)\|_1$ dominates the above expression for large t. By assuming H is row-compressible, we can derive a bound on $\|\mu(E)\|_1$.

Lemma 4. *Assume H is p-row-compressible with magnitude R, where $H = Z + E$ for s-row-sparse Z and arbitrary E. Then, $\|\mu(E)\|_1 \leq R \cdot \frac{s^{1-\frac{1}{p}}}{\frac{1}{p}-1}$.*

Lemma 4 can then be combined with Theorem 3 to bound the error due to pruning via i-SpaSP.

Theorem 4. *Assume Algorithm 4 is run for a sufficiently large number of iterations t. If Assumption 2 holds and H is p-row-compressible with factor R, the output residual between the dense network and the pruned network discovered via i-SpaSP can be bounded as follows:*

$$\|V_t\|_F \leq \mathcal{O}\left(\frac{s^{\frac{1}{2}-\frac{1}{p}} p(2\sqrt{s}+1)}{1-p} \right).$$

Proof. This follows directly from substituting Lemma 4 into Theorem 3, assuming t is large enough such that $(0.444)^t \|W^{(1)}\|_F \|\mu(H)\|_2 \approx 0$, and factoring out constants in the resulting expression. □

Theorem 4 indicates that the quality of the pruned network is dependent upon s and p. Intuitively, one would expect that lower values of p (implying sparser H) would make pruning easier, as neurons corresponding to zero rows in H could be eliminated without consequence. This trend is observed exactly within Theorem 4; e.g., for $p = \{\frac{3}{4}, \frac{1}{2}, \frac{1}{4}\}$ we have $\|V_t\|_F \leq \{\mathcal{O}(s^{-\frac{1}{3}}), \mathcal{O}(s^{-1}), \mathcal{O}(s^{-3})\}$, respectively. *To the best of the authors' knowledge, our work is the first to theoretically characterize pruning error with respect to sparsity properties of network hidden representations.* This bound can also be extended to similarly-structured (see Appendix), multi-layer networks:

Theorem 5. *Consider an L-hidden-layer network with weight matrices $\{W^{(0)}, \dots, W^{(L)}\}$ and hidden representations $\{H^{(1)}, \dots H^{(L)}\}$. The hidden representations of each layer are assumed to have dimension d for simplicity. We define $H^{(\ell)} = \sigma(W^{(\ell)} \cdot H^{(\ell-1)})$, where $H^{(1)} = \sigma(W^{(0)} \cdot X)$ and $H^{(L)} = W^{(L)} \cdot H^{(L-1)}$. We assume all weight matrices other than $W^{(0)}$ obey Assumption 2 and all hidden representations other than $H^{(L)}$ are p-row-compressible. i-SpaSP is applied greedily to prune each network layer, in layer order, from d to s hidden neurons. Given sufficient iterations t, the residual between pruned and dense multi-layer network output behaves as:*

$$\|V_t^{(L)}\|_F \leq \mathcal{O}\left(\sum_{i=1}^{L} \left(14 + \frac{7}{\sqrt{s}}\right)^{L-i+1} \left(\|W^{(L)}\|_F \prod_{j=1}^{L-i} \|vec(W^{(j)})\|_1 \right) \left(\frac{d^{\frac{L-i}{2}} s^{1-\frac{1}{p}}}{\frac{1}{p}-1} \right) \right) \tag{18}$$

Pruning error in (18) is summed over each network layer. Considering layer i, the green factor is inherited from Theorem 3 with an added exponent due to a recursion over network layers after i. Similarly, the blue

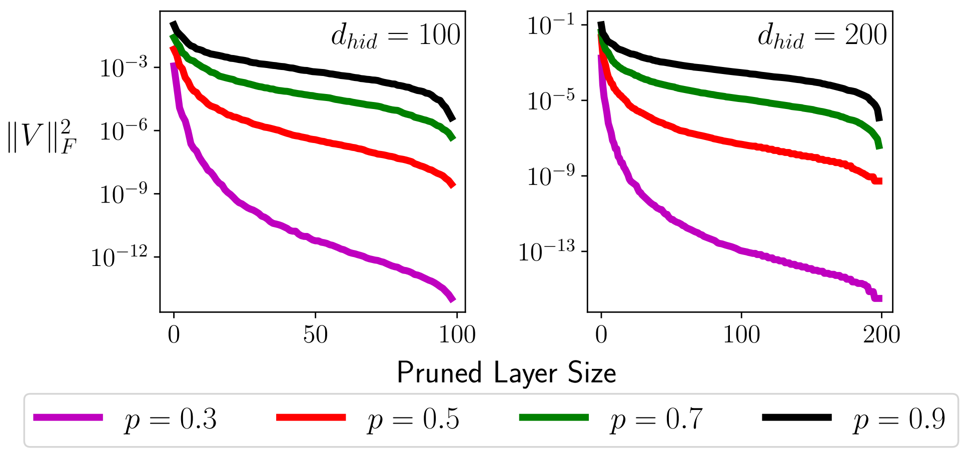

Figure 8: i-SpaSP pruning experiments for two-layer networks of different sizes and p ratios.

factor accounts for propagation of error through weight matrices after layer i, *revealing that green and blue factors account for propagation of error through network layers*. The red portion of (18), which captures the convergence properties of multi-layer pruning, comes from Lemma 4, where an extra factor $d^{\frac{L-i}{2}}$ arises as an artifact of the proof.[20] Because d is fixed multiple of s determined by the pruning ratio, this factor disappears when p is small, leading the expression to converge. For example, if $p = \frac{1}{4}$, the red expression in (18) behaves asymptotically as $\{\mathcal{O}(s^{-2.5}), \mathcal{O}(s^{-2}), \mathcal{O}(s^{-1.5}), \dots\}$ for layers at the end of the network moving backwards.

4.6 Experiments

Within this section, we provide empirical analysis of i-SpaSP. We first present synthetic results using two-layer neural networks to numerically verify Theorem 4. Then, we perform experiments with two-layer networks on MNIST [52], convolutional neural networks (CNNs) on ImageNet (ILSVRC2012), and multi-lingual BERT (mBERT) [53] on the cross-lingual NLI corpus (XNLI) [46]. For all experiments, we adopt best practices from previous work [162] to determine pruning ratios within the dense network, often performing less pruning on sensitive layers; see the Appendix for details. As baselines, we adopt both greedy selection methodologies [251, 252] and several common, heuristic methods. We find that, in addition to improving upon the pruning efficiency of GFS (i.e., see Section 4.4.2 for runtime comparisons on ResNet pruning experiments), i-SpaSP performs comparably to baselines in all cases, demonstrating that it is both performant and efficient.

4.6.1 Synthetic Experiments

To numerically verify Theorem 4, we construct synthetic H matrices with different row-compressibility ratios p, denoted as $\mathcal{H} = \{H^{(i)}\}_{i=1}^{K}$. For each $p \in \{0.3, 0.5, 0.7, 0.9\}$, we randomly generate three unique entries

[20]This artifact arises because ℓ_1 norms must be replaced with ℓ_2 norms in certain areas to form a recursion over network layers. This artifact can likely be removed by leveraging sparse matrix analysis for expander graphs [13], but we leave this as future work to simplify the analysis.

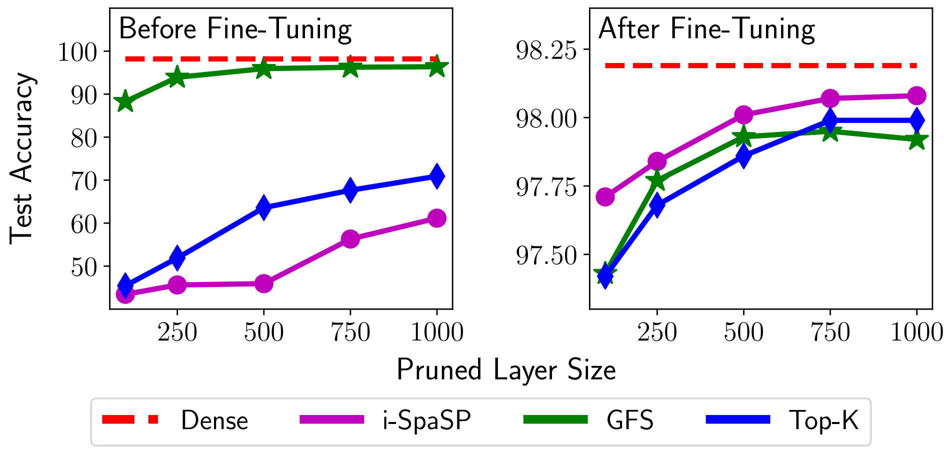

Figure 9: Performance of networks pruned with different greedy algorithms on MNIST before (left) and after (right) fine-tuning.

within $\mathcal{H}$ (i.e., $K = 12$) and present average performance across them all. We prune a randomly-initialized $W^{(1)}$ from size $d_{hid} \times d_{out}$ to sizes $s \times d_{out}$ for $s \in [d_{hid}]$. (i.e., each setting of s is a separate experiment) using $T = 20$.[21]. We test $d_{hid} \in \{100, 200\}$, and the same $W^{(1)}$ matrix is used for experiments with equal hidden dimension; see the Appendix for more details.

Results are displayed in Figure 8, where $\|V\|_F^2$ is shown to decay polynomially with respect to the number of neurons within the pruned network s. Furthermore, as predicted by Theorem 4, the decay rate of $\|V\|_F^2$ increases as p decreases, revealing that higher levels of sparsity within the dense network's hidden representations improve pruning performance and speed with respect to s. This trend holds for all hidden dimensions that were considered.

4.6.2 Two-Layer Networks

We perform pruning experiments with two-layer networks on MNIST [52]. All MNIST images are flattened and no data augmentation is used. The dense network has 10×10^3 hidden neurons and is pre-trained before pruning. We prune the dense network using i-SpaSP, GFS, and Top-K, which we use as a naive greedy selection baseline.[22] After pruning, the network is fine-tuned using stochastic gradient descent (SGD) with momentum. Performance is reported as an average across three separate trails; see the Appendix for more details.

As shown in Figure 9-right, i-SpaSP outperforms other pruning methodologies after fine-tuning in all experimental settings. Networks obtained with GFS perform well without fine-tuning (i.e., Figure 9-left) because neurons are selected to minimize loss during the pruning process. Because i-SpaSP selects neurons based upon the importance criteria described in Section 4.4, the pruning process does not directly minimize

[21]We find that the active set of neurons selected by i-SpaSP becomes stable (i.e., few neurons are modified) after 20 iterations or less in all synthetic experiments.

[22]Top-k selects k neurons with the largest-magnitude hidden representations in a mini-batch of data. It is a naive baseline for greedy selection that is not used in previous work to the best of our knowledge.

	Pruning Method	Top-1 Acc.	FLOPS
ResNet34	Full Model [106]	73.4	3.68G
	Filter Pruning [162]	72.1	2.79G
	Rethinking Pruning [163]	72.0	2.79G
	More is Less [55]	73.0	2.75G
	i-SpaSP	73.5	2.69G
	GFS [251]	73.5	2.64G
	Multi-Layer GFS [252]	73.5	2.20G
	SFP [107]	71.8	2.17G
	FPGM [108]	72.5	2.16G
	i-SpaSP	72.5	2.13G
	GFS [251]	72.9	2.07G
	Multi-Layer GFS [252]	73.3	1.90G
MobileNetV2	Full Model [207]	72.0	314M
	i-SpaSP	71.6	260M
	GFS [251]	71.9	258M
	Multi-Layer GFS [252]	72.2	245M
	i-SpaSP	71.3	242M
	LEGR [44]	71.4	224M
	Uniform	70.0	220M
	AMC [109]	70.8	220M
	i-SpaSP	70.7	220M
	GFS [251]	71.6	220M
	Multi-Layer GFS [252]	71.7	218M
	Meta Pruning [161]	71.2	217M

Table 7: Test accuracy of ResNet34 and MobileNetV2 models pruned to different FLOP levels with various pruning algorithms on ImageNet.

training loss, thus leading to poorer performance prior to fine-tuning (i.e., Top-K exhibits similar behavior). Nonetheless, i-SpaSP, in addition to improving upon the pruning efficiency of GFS, discovers a set of neurons that more closely recovers dense network output, as revealed by its superior performance after fine-tuning.

4.6.3 Deep Convolutional Networks

We perform structured filter pruning of ResNet34 [106] and MobileNetV2 [207] with i-SpaSP on ImageNet (ILSVRC 2012). Beginning with public, pre-trained models [193], we use i-SpaSP to prune chosen convolutional blocks within each network, then fine-tune the pruned model using SGD with momentum. After pruning each block, we perform a small amount of fine-tuning; see the Appendix for more details. Numerous heuristic and greedy selection-based algorithms are adopted as baselines; see Table 7.

ResNet34. We prune ResNet34 to 2.69 and 2.13 GFlops with i-SpaSP. As shown in Table 7, i-SpaSP yields comparable performance to GFS variants at similar FLOP levels; e.g., i-SpaSP matches 75.5% test accuracy of

Method	Pruning Ratio	Top-1 Acc.
Full Model	-	71.02
Uniform	25%	62.73
	40%	66.43
[177]	25%	62.97
	40%	67.31
i-SpaSP	25%	63.70
	40%	68.11

Table 8: Test accuracy of mBERT models pruned and fine-tuned on the XNLI dataset.

GFS with the 2.69 GFlop model and performs within 1% of GFS variants for the 2.13 GFlop model. However, the multi-layer variant of GFS [252] does discover sub-networks with fewer FLOPS and similar performance in both cases. i-SpaSP improves upon or matches the performance of all heuristic methods at similar FLOP levels.

MobileNetV2. We prune MobileNetV2 to 260, 242, and 220 MFlops with i-SpaSP. In all cases, sub-networks discovered with i-SpaSP achieve performance within 1% of those obtained via both GFS variants and heuristic methods. Although MobileNetV2 performance is relatively lower in comparison to ResNet34, i-SpaSP is still capable of pruning the more difficult network to various different FLOP levels and performs comparably to baselines in all cases.

Discussion. Within Table 7, the performance of i-SpaSP is never more than 1% below that of a similar-FLOP model obtained with a baseline pruning methodology, including both greedy selection and heuristic-based methods. As such, similar to GFS, i-SpaSP can be seen as a theoretically-grounded pruning methodology that is practically useful, even in large-scale experiments. Such pruning methodologies that are both theoretically and practically relevant are few. In comparison to GFS variants, i-SpaSP significantly improves pruning efficiency; see Section 4.4.2. Thus, i-SpaSP can be seen as a viable alternative to GFS—both theoretically and practically—that may be preferable when runtime is a major concern.

4.6.4 Transformer Networks

We perform structured pruning of attention heads using the mBERT model [53] on the XNLI dataset [46], which contains textual entailment annotations across 15 different languages. We begin with a pre-trained mBERT model and fine-tune it on XNLI prior to pruning. Then, structured pruning is performed on the attention heads of each layer using i-SpaSP, uniform pruning (i.e., randomly removing a fixed ratio of attention heads), and a sensitivity-based masking approach [177] (i.e., a state-of-the-art heuristic approach for structured attention head pruning for transformers). After pruning models to a fixed ratio of 25% or 40% of original attention heads, we fine-tune the pruned networks and record their performance on the XNLI test set; see the Appendix for further details.

As shown in Table 8, models pruned with i-SpaSP outperform those pruned to the same ratio with either uniform or heuristic pruning methods in all cases. The ability of i-SpaSP to effectively prune transformers demonstrates that the methodology can be applied to the structured pruning of numerous different architectures without significant implementation changes (i.e., using the automatic differentiation approach described in Section 4.4.1). Furthermore, i-SpaSP even outperforms a state-of-the-art heuristic approach for the structured pruning of transformer attention heads, thus again highlighting the strong empirical performance of i-SpaSP in comparison to heuristic pruning methods that lack theoretical guarantees.

4.7 Conclusion

We propose i-SpaSP, a pruning methodology for neural networks inspired by sparse signal recovery. Our methodology comes with theoretical guarantees that indicate, for both two and multi-layer networks, the quality of a pruned network decays polynomially with respect to its size. We connect this theoretical analysis to properties of the dense network, showing that pruning performance improves as the dense network's hidden representations become more sparse. Practically, i-SpaSP performs comparably to numerous baseline pruning methodologies in large-scale experiments and drastically improves upon the computationally efficiency of the most common provable pruning methodologies. As such, i-SpaSP is a practical, provable, and efficient algorithm that we hope will enable a better understanding of neural network pruning both in theory and practice. In future work, we wish to extend i-SpaSP to cover cases in which network pruning and training are combined into a single process, such as utilizing regularization-based approaches to induce sparsity during pre-training or updating network weights to improve sub-network performance as pruning occurs.

5 Cold Start Streaming Learning for Deep Networks

The ability to dynamically adapt neural networks to newly-available data without performance deterioration would revolutionize deep learning applications. Streaming learning (i.e., learning from one data example at a time) has the potential to enable such real-time adaptation, but current approaches i) freeze a majority of network parameters during streaming and ii) are dependent upon offline, base initialization procedures over large subsets of data, which damages performance and limits applicability. To mitigate these shortcomings, we propose Cold Start Streaming Learning (CSSL), a simple, end-to-end approach for streaming learning with deep networks that uses a combination of replay and data augmentation to avoid catastrophic forgetting.

Because CSSL updates all model parameters during streaming, the algorithm is capable of beginning streaming from a random initialization, making base initialization optional. Going further, the algorithm's simplicity allows theoretical convergence guarantees to be derived using analysis of the Neural Tangent Random Feature (NTRF). In experiments, we find that CSSL outperforms existing baselines for streaming learning in experiments on CIFAR100, ImageNet, and Core50 datasets. Additionally, we propose a novel multi-task streaming learning setting and show that CSSL performs favorably in this domain. Put simply, CSSL performs well and demonstrates that the complicated, multi-step training pipelines adopted by most streaming methodologies can be replaced with a simple, end-to-end learning approach without sacrificing performance.

5.1 Introduction

Many autonomous applications would benefit from real-time, dynamic adaption of models to new data. As such, online learning[23] has become a popular topic in deep learning; e.g., continual [165, 263], lifelong [6, 33], incremental [30, 202], and streaming [102, 103] learning. However, the (potentially) non-i.i.d. nature of incoming data causes catastrophic forgetting [143, 175], thus complicating the learning process.

Batch-incremental learning [30, 202], where batches of data—typically sampled from disjoint sets of classes or tasks within a dataset—become available to the model sequentially, is a widely-studied form of online learning. Within this setup, however, one must wait for a sizeable batch of data[24] to accumulate before the model is updated with an expensive, offline training procedure over new data, thus introducing latency that prevents real-time model updates.

To avoid such latency, we adopt the streaming learning setting where i) each data example is seen once and ii) the dataset is learned in a single pass [102]. Streaming learning performs brief, online updates (i.e., one or a few forward/backward passes) for each new data example, *forcing learning of new data to occur in real-time.* Additionally, streaming learning techniques can be adapted to cope with batches of data instead of single examples [237], while batch-incremental learning techniques tend to deteriorate drastically given smaller batch sizes [103]. As such, the streaming learning setting, which has recently been explored for applications with deep neural networks [102, 103, 104], is generic and has the potential to enable low-cost updates of deep networks to incoming data.

Current streaming methodologies for deep neural networks learn in two-phases: base initialization and streaming. During base initialization, network parameters (and other modules, if needed) are pre-trained over a subset of data. Then, a majority of network parameters are frozen (i.e., not updated) throughout the learning process. During streaming, most approaches maintain a replay buffer—though other techniques exist [104]—to avoid catastrophic forgetting by allowing prior data to be sampled and included in each online update.

The current, multi-stage streaming learning pipeline suffers a few notable drawbacks. Namely, the learning process is dependent upon a latency-inducing, offline pre-training procedure and a majority of network parameters are not updated during the streaming process. As a result, the underlying network i) has reduced representational capacity, ii) cannot adapt a large portion of its parameters to new data, and iii) is dependent upon high-quality pre-training (during base initialization) to perform well. Given these considerations, one may begin to wonder whether a simpler streaming procedure could be derived to realize the goal of adapting deep networks to new data in real time.

Our Proposal. Inspired by the simplicity of offline training, we propose a novel method for streaming learning with deep networks, called Cold Start Streaming Learning (CSSL), that updates all network parameters in an end-to-end fashion throughout the learning process. Because no parameters are fixed by CSSL, base initialization and pre-training procedures are optional—streaming can begin from a completely random initialization (hence, "cold start" streaming learning). By leveraging a basic replay mechanism coupled with

[23]We use "online learning" to generically describe methodologies that perform training in a sequential manner.

[24]These "batches" are large (e.g., a 100-class subset of ImageNet with $>$100,000 data examples [202]) and using smaller batches damages performance [22, 103].

sophisticated data augmentation techniques, CSSL outperforms existing streaming techniques in a variety of domains. A summary of our contributions is as follows:

- We propose CSSL, a simple streaming methodology that combines replay buffers with sophisticated data augmentation, and provide extensive comparison to existing techniques on common class-incremental streaming problems. We show that CSSL often outperforms baseline methodologies by a large margin.
- We leverage techniques related to neural tangent random feature (NTRF) analysis [29] to prove a theoretical bound on the generalization loss of CSSL over streaming iterations.
- We propose a multi-task streaming learning benchmark, where multiple, disjoint classification datasets are presented to the model sequentially and in a streaming fashion. We show that CSSL enables significant performance improvements over baseline methodologies in this new domain.
- We extensively analyze models trained via CSSL, showing that resulting models $i)$ achieve impressive streaming performance even when beginning from a random initialization (i.e., a cold start); $ii)$ are robust to the compression of examples in the replay buffer for reduced memory overhead; and $iii)$ provide highly-calibrated confidence scores due to our proposed data augmentation policy.

5.2 Related Work

Online Learning. Numerous experimental setups have been considered for online learning, but they all share two properties: $i)$ the sequential nature of the training process and $ii)$ performance deterioration due to catastrophic forgetting when incoming data is non-i.i.d. [137, 175]. Replay mechanisms, which maintain a buffer of previously-observed data (or a generative model to produce such data [201, 211]) to include in online updates, are highly-effective at preventing catastrophic forgetting at scale [34, 57, 103], leading us to base our proposed methodology upon replay. Similarly, knowledge distillation [114] can prevent performance deterioration by stabilizing feature representations throughout the online learning process [119, 245], even while training the network end-to-end (e.g., end-to-end incremental learning [30] and iCarl [202]). Though distillation and replay are widely-used, numerous other approaches to online learning also exist (e.g., architectural modification [58, 206], regularization [54, 153], and dual memory [20, 136]).

Streaming Learning. Streaming, which we study in this work, performs a single pass over the dataset, observing each sample once [102]. Having recently become popular in deep learning, streaming learning [2, 80, 103, 104] trains the model in two-phases: base initialization and streaming. Base initialization uses a subset of data to pre-train the model and initialize relevant network modules. Then, the streaming phase learns the dataset in a single-pass with a majority of network parameters fixed. Within this training paradigm, replay-based techniques perform well at scale [103], though other approaches may also be effective [104].

Data Augmentation. The success of the proposed methodology is enabled by our data augmentation policy. Though data augmentation for computer vision is well-studied [212], we focus upon interpolation methods and learned augmentation policies. Interpolation methods (e.g., Mixup [125, 240, 264] and CutMix [259]) take stochastically-weighted combinations of images and label pairs during training, which provides regularization benefits. Learned augmentation policies (e.g., AutoAugment [48, 98, 157] and RandAugment [49]), on the other hand, consider a wide scope of augmentation techniques and adopt a data-centric approach to learn an optimal augmentation policy (i.e., using reinforcement learning or gradient-based techniques).

Confidence Calibration. Beyond the core methodology of CSSL, we extensively explore the confidence calibration properties of resulting models. Put simply, confidence calibration is the ability of a model to accurately predict the probability of its own correctness—poor predictions should be made with low confidence and vice versa. Numerous methodologies have been proposed for producing calibrated models, including post-hoc softmax temperature scaling [90], ensemble-based techniques [146], Mixup [225, 264], Monte-Carlo dropout, and uncertainty estimates in Bayesian networks [78, 183]. Confidence calibration has also been applied to problems including domain shift and out-of-distribution detection due to its ability to filter incorrect, or low confidence predictions [111, 112].

Multi-task learning. *Our work is the first to consider multi-task streaming learning*, though multi-task learning has been previously explored for both offline and batch-incremental settings. [118] studies the sequential learning of several datasets via knowledge distillation and replay, and several other works study the related problem of domain shift [77, 131], where two datasets are learned in a sequential fashion. For the offline setting, multi-task learning has been explored extensively, leading to a variety of successful learning methodologies for computer vision [167], natural language processing [222], multi-modal learning [187, 241], and more. The scope of work in offline multi-task learning is too broad to provide a comprehensive summary here, though numerous surveys on the topic are available [205, 242].

5.3 Methodology

The proposed streaming methodology, formulated in Algorithm 5, begins either from pre-trained parameters or a random-initialization (`Initialize` in Algorithm 5) and utilizes a replay buffer $\mathcal{R}$ to prevent catastrophic forgetting. At each iteration, CSSL receives a new data example $x_t, y_t := \mathcal{D}_t$, combines this data with random samples from the replay buffer, performs a stochastic gradient descent (SGD) update with the combined data, and stores the new example within the buffer for later replay. Within this section, we will first provide relevant preliminaries and definitions, then each component of CSSL will be detailed and contrasted with prior work.

5.3.1 Preliminaries

Problem Setting. Following prior work [103], we consider the problem of streaming for image classification.[25] In most cases, we perform class-incremental streaming experiments, in which examples from each distinct semantic class within the dataset are presented to the learner sequentially. However, experiments are also performed using other data orderings; see Section 5.6.2. Within our theoretical analysis only, we consider a binary classification problem that maps vector inputs to binary labels; see Section 5.5 for further details. Such a problem setup is inspired by prior work [9, 29].

Streaming Learning Definition. Streaming learning considers an incoming data stream $\mathcal{D} = \{(x_t, y_t)\}_{t=1}^{n}$[26] (i.e., t denotes temporal ordering) and adopts the following rule set during training:

[25]Streaming learning has been considered in the object detection domain [2], but we consider this application beyond the scope of our work.

[26]E.g., $x_t \in \mathbb{R}^{3\times H\times W}$ (RGB image) and $y_t \in \mathbb{Z}^{\geq 0}$ (class label) for computer vision applications.

Algorithm 5: Cold Start Streaming Learning

```
Parameters: $\mathbf{W}$ model parameters, $\mathcal{D}$ data stream, $\mathcal{R}$ replay buffer, $C$ maximum replay buffer size,
  $B$ number of replay samples per iteration

# Initialize model parameters randomly or via pre-training (if possible)
$\mathbf{W} :=$ Initialize()
$\mathcal{R} := \emptyset$
for $t = 1, \ldots, |\mathcal{D}|$ do
    # Sample data from the replay buffer to combine with new data from $\mathcal{D}$
    $x_{\text{new}}, y_{\text{new}} := \mathcal{D}_t$
    $\mathcal{X}_{\text{replay}}, \mathcal{Y}_{\text{replay}} :=$ ReplaySample($\mathcal{R}, B$)
    $\mathcal{X}, \mathcal{Y} := \{x_{\text{new}}\} \cup \mathcal{X}_{\text{replay}}, \{y_{\text{new}}\} \cup \mathcal{Y}_{\text{replay}}$

    # Update all model parameters over augmented new and replayed data
    StreamingUpdate($\mathbf{W}$, Augment($\mathcal{X}, \mathcal{Y}$))

    # Compress and Store the new data example for replay
    ReplayStore($\mathcal{R}$, (Compress($x_{\text{new}}$), $y_{\text{new}}$))

    # Evict data from the replay buffer to maintain capacity
    if $|\mathcal{R}| > C$ then
        ReplayEvict($\mathcal{R}$)
    end
end
```

1. Each unique data example appears once in $\mathcal{D}$.
2. The order of data within $\mathcal{D}$ is arbitrary and possibly non-iid.
3. Model evaluation may occur at any time t.

Notably, these requirements make no assumptions about model state prior to the commencement of the streaming process. Previous methodologies perform offline, base initialization procedures before streaming begins, while CSSL may either begin streaming from randomly-initialized or pre-trained parameters. As such, CSSL is capable of performing streaming even without first observing examples from $\mathcal{D}$, while other methodologies are reliant upon data-dependent base initialization.

Evaluation Metric. In most experiments, performance is measured using Ω_{all} [102], defined as $\Omega_{\text{all}} = \frac{1}{T}\sum_{t=1}^{T}\frac{\alpha_t}{\alpha_{\text{offline},t}}$, where α_t is streaming performance at testing event t, $\alpha_{\text{offline},t}$ is offline performance at testing event t, and T is the number of total testing events. Ω_{all} captures aggregate streaming performance (relative to offline training) throughout streaming. A higher score indicates better performance.

As an example, consider a class-incremental streaming setup with two testing events: one after ½ of the classes have been observed and one at the end of streaming. The streaming learner is evaluated at each testing event, yielding accuracy α_1 and α_2. Two models are trained offline over ½ of the classes and the full dataset, respectively, then evaluated to yield $\alpha_{\text{offline},1}$ and $\alpha_{\text{offline},2}$. From here, Ω_{all} is given by $\frac{1}{2}\left(\frac{\alpha_1}{\alpha_{\text{offline},1}} + \frac{\alpha_2}{\alpha_{\text{offline},2}}\right)$.

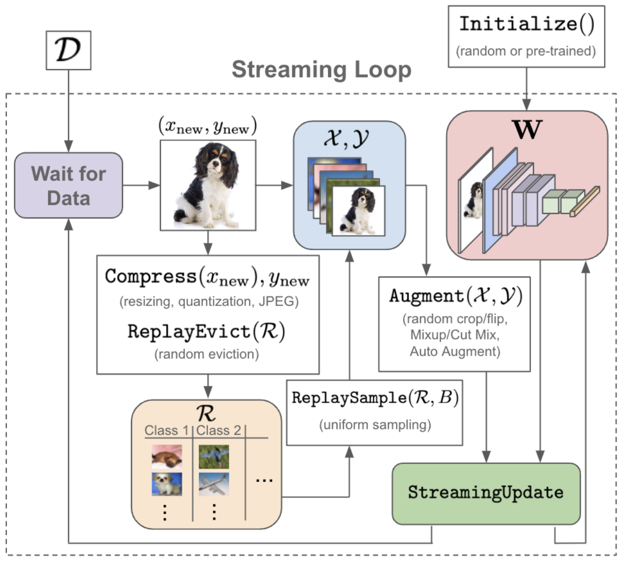
$\mathcal{D}$
Initialize()
(random or pre-trained)
Streaming Loop
Wait for Data
$(x_{\text{new}}, y_{\text{new}})$
$\mathcal{X}, \mathcal{Y}$
W
Compress$(x_{\text{new}}), y_{\text{new}}$
(resizing, quantization, JPEG)
ReplayEvict$(\mathcal{R})$
(random eviction)
Augment$(\mathcal{X}, \mathcal{Y})$
(random crop/flip, Mixup/Cut Mix, Auto Augment)
ReplaySample$(\mathcal{R}, B)$
(uniform sampling)
$\mathcal{R}$
Class 1
Class 2
StreamingUpdate

Figure 10: Illustration of CSSL following the notation of Algorithm 5.

5.3.2 Cold Start Streaming Learning

Replay Buffer. As the learner progresses through the data stream, full images and their associated labels are stored (`ReplayStore`) within a fixed-size replay buffer $\mathcal{R}$. The replay buffer $\mathcal{R}$ has a fixed capacity C. Data can be freely added to $\mathcal{R}$ if $|\mathcal{R}| < C$, but eviction must occur (`ReplayEvict` in Algorithm 5) to make space for new examples when the buffer is at capacity. One of the simplest eviction policies—which is adopted in prior work [103]—is to $i)$ identify the class containing the most examples in the replay buffer and $ii)$ remove a random example from this class. This policy, which we adopt in CSSL, is computationally efficient and performs similarly to more sophisticated techniques; see Appendix D.2.3.

Data Compression. Data is compressed (`Compress` in Algorithm 5) before addition to $\mathcal{R}$. By freezing a majority of network layers during streaming, previous methods can leverage learned quantization modules and pre-trained feature representations to drastically reduce the size of replay data [103, 104]. For CSSL, full images, which induce a larger memory overhead, are stored in $\mathcal{R}$ and no network parameters are fixed, meaning that `Compress` cannot rely on pre-trained, fixed network layers.

We explore several, data-independent `Compress` operations, such as resizing images, quantizing the integer representations of image pixels, and storing images on disk with JPEG compression; see Appendix D.2.2. Such compression methodologies significantly reduce memory overhead without impacting performance. Because storing the replay buffer on disk is not always possible (e.g., edge devices may lack disk-based storage), we present this as a supplemental approach and always present additional results without JPEG compression.

Due to storing full images in the replay buffer, the proposed methodology has more memory overhead relative to prior techniques, making it less appropriate for memory-limited scenarios. However, many streaming applications (e.g., e-commerce recommendation systems, pre-labeling for data annotation, etc.) store and update models on cloud servers, making memory efficiency less of a concern. *We recommend CSSL*

No. Base Init.Classes	Top-1 Ω_{all}	Top-5 Ω_{all}
10	0.478	0.651
50	0.727	0.848
100	0.835	0.856

Table 9: Ω_{all} of REMIND on ImageNet with different amounts of base initialization data.

for such scenarios, as it outperforms prior methods given sufficient memory for replay.

Model Updates. For each new example within $\mathcal{D}$, CSSL updates learner parameters Θ (`StreamingUpdate` in Algorithm 5) using the new example and N replay buffer samples obtained via `ReplaySample`. Similar to prior work, `ReplaySample` uniformly samples data from $\mathcal{R}$—alternative sampling strategies provide little benefit at scale [5, 7, 103]; see Appendix D.2.4. `StreamingUpdate` is implemented as a simple SGD update of all network parameters over new and replayed data.

Streaming updates pass the mixture of new and replayed data through a data augmentation pipeline (`Augment` in Algorithm 5). Though prior work uses simple augmentations [30, 103, 228], we explore data interpolation [259, 264] and learned augmentation policies [48]. In particular, CSSL combines random crops and flips, Mixup, Cutmix, and Autoaugment into a single, sequential augmentation policy[27]; see Appendix D.2.1 for further details. *Curating a high quality data augmentation pipeline is the key CSSL's impressive performance*.

Our Methodology. In summary, CSSL, illustrated in Figure 10, initializes the network either randomly or with pre-trained parameters (`Initialize`). Then, for each new data example, N examples are sampled uniformly from the replay buffer (`ReplaySample`), combined with the new data, passed through a data augmentation pipeline (`Augment`), and then used to perform a training iteration (`StreamingUpdate`). Each new data example is added to the replay buffer (`ReplayStore`), and an example may be randomly removed from the replay buffer via random, uniform selection (`ReplayEvict`) if the capacity C is exceeded.

5.4 Why is this useful?

The proposed methodology has two major benefits:

- **Full Plasticity** (i.e., no fixed parameters)
- **Pre-Training is Optional**

Such benefits are unique to CSSL; see Appendix D.3. However, one may question the validity of these "benefits"—*do they provide any tangible value?*

Full Plasticity. CSSL updates all model parameters throughout the streaming process.[28] Some incremental learning methodologies train networks end-to-end [30, 119, 245], but these approaches either i) cannot be

[27]Mixup and Cutmix are not performed simultaneously. Our policy randomly chooses between Mixup or Cutmix for each data example with equal probability.

[28]Training the network end-to-end within CSSL does make model inference and updates more computationally expensive, which adds latency to streaming updates; see Appendix D.2.7 for further analysis.

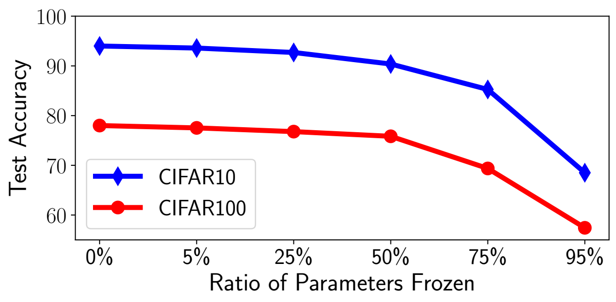

Figure 11: Test acccuracy of ResNet18 models that are pre-trained on ImageNet and fine-tuned on CIFAR10/100 with different ratios of frozen parameters.

applied or ii) perform poorly when adapted to the streaming domain [103]; see Appendix D.2.6. Fixing network parameters is detrimental to the learning process. To show this, we pre-train a ResNet18 on ImageNet and fix different ratios of network parameters during fine-tuning on CIFAR10/100, finding that final accuracy deteriorates monotonically with respect to the ratio of frozen parameters; see Figure 11. Even given high-quality pre-training, fixing network parameters i) limits representational capacity and ii) prevents network representations from being adapted to new data, making end-to-end training a more favorable approach.

Pre-Training is Optional. Prior streaming methods perform base initialization (i.e., fitting of network parameters and other modules over a subset of data) prior to streaming. Not only does base initialization introduce an expensive, offline training procedure, but it makes model performance dependent upon the availability of sufficient (possibly unlabeled [80]) pre-training data prior to streaming. Streaming performance degrades with less base initialization data; see Table 9. Thus, if little (or worse, no) data is available when streaming begins, such methods are doomed to poor performance.

CSSL has no dependency upon pre-training because the network is trained end-to-end during streaming. Thus, the CSSL training pipeline is quite simple—*just initialize, then train*. Such simplicity makes CSSL easy to implement and deploy in practice. We emphasize that CSSL *can* begin streaming from a pre-trained set of model parameters, which often leads to better performance. The benefit of CSSL, however, is that such a pre-training step is *completely optional* and network parameters are still updated during streaming.

Implications and Potential Applications. The proposed methodology yields improvements in *performance*, *simplicity*, and *applicability* relative to prior work. Fixing no network parameters enables better *performance* by increasing network representational capacity and adaptability, while the ability to stream from a cold start *simplifies* the streaming pipeline and makes streaming *applicable* to scenarios that lack sufficient data for base initialization.

Consider, for example, pre-labeling for data annotation, where a streaming learner works with a human annotator to improve labeling efficiency. In this case, the annotated dataset may be built from scratch, meaning that no data is yet available when streaming begins. Similarly, cold-start scenarios for recommendation or classification systems often lack prior data from which to learn. Though pre-trained models may sometimes be downloaded online, this is not possible for domains that align poorly with public models and datasets (e.g., medical imaging). While prior approaches fail in these scenarios due to a dependence upon high-quality base initialization, CSSL can simply begin streaming from a random initialization and still perform well.

5.5 Theoretical Results

CSSL's simple training pipeline enables the algorithm to be analyzed theoretically. To do this, we extend analysis of the neural tangent random feature (NTRF) [29] to encompass multi-layer networks trained via streaming to perform binary classification using a replay mechanism and data augmentation. The analytical setup is similar to Algorithm 5 with a few minor differences. We begin by providing relevant notation and definitions, then present the main theoretical result.

5.5.1 Preliminaries and Definitions

Notation. We denote scalars and vectors with lower case letters, distinguished by context, and matrices with upper case letters. We define $[l] = \{1, 2, \dots, l\}$ and $\|v\|_p = (\sum_{i=1}^{d} |v_i|^p)^{\frac{1}{p}}$ for $v = (v_1, v_2, \dots, v_d)^\top \in \mathbb{R}^d$ and $1 \leq p < \infty$. We also define $\|v\|_\infty = \max_i |v_i|$. For matrix $A \in \mathbb{R}^{m \times n}$, we use $\|A\|_0$ to denote the number of non-zero entries in A. We use $i \sim \mathcal{N}$ to denote a random sample i from distribution $\mathcal{N}$, and $\mathcal{N}(\mu, \sigma)$ to denote the normal distribution with mean μ and standard deviation σ. $\mathbb{1}\{\cdot\}$ is the indicator function. We then define $\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$ and $\|A\|_p = \max_{\|v\|_p = 1} \|Av\|_p$ for $p \geq 1$ and $v \in \mathbb{R}^n$. For two matrices $A, B \in \mathbb{R}^{m \times n}$, we have $\langle A, B \rangle = \text{Tr}(A^\top B)$. For vector $v \in \mathbb{R}^d$, $\texttt{diag}(v) \in \mathbb{R}^{d \times d}$ is a diagonal matrix with the entries of v on the diagonal. We also define the asymptotic notations $\mathcal{O}(\cdot)$ and $\Omega(\cdot)$ as follows. If a_n and b_n are two sequences, $a_n = \mathcal{O}(b_n)$ if $\limsup_{n \to \infty} |\frac{a_n}{b_n}| < \infty$ and $a_n = \Omega(b_n)$ if $\limsup_{n \to \infty} |\frac{a_n}{b_n}| > 0$.

Network Formulation. We consider fully forward neural networks with width m, depth L, input dimension d, and an output dimension of one. The weight matrices of the network are $W_1 \in \mathbb{R}^{m \times d}$, $W_\ell \in \mathbb{R}^{m \times m}$ for $\ell = 2, \dots, L-1$, and $W_L \in \mathbb{R}^{1 \times m}$.[29] We denote $\mathbf{W} = \{W_1, \dots, W_L\}$ and define $\langle \mathbf{W}, \mathbf{W}' \rangle = \sum_{i=1}^{L} \langle W_i, W_i' \rangle$ when $\mathbf{W}, \mathbf{W}'$ are two sets containing corresponding weight matrices with equal dimension. All of such sets with corresponding weight matrices of the same dimension form the set $\mathcal{W}$. Then, a forward pass with weights $\mathbf{W} = \{W_1, \dots, W_L\} \in \mathcal{W}$ for input $x \in \mathbb{R}^d$ is given by:

$$f_{\mathbf{W}}(x) = \sqrt{m} \cdot W_L \sigma(W_{L-1} \sigma(W_{L-2} \dots \sigma(W_1(x)) \dots)), \tag{19}$$

where $\sigma(\cdot)$ is the ReLU activation function. Following [29], the first and last weight matrices are not updated during training.

Objective Function. For data stream $\mathcal{D} = \{(x_t, y_t)\}_{t=1}^{n}$, we define $L_{\mathcal{D}}(\mathbf{W}) \triangleq \mathbb{E}_{(x,y) \sim \mathcal{D}} L_{(x,y,\xi)}(\mathbf{W})$, where $L_{(x,y,\xi)}(\mathbf{W}) = \ell[y \cdot f_{\mathbf{W}}(x + \xi)]$ is the loss over $(x, y) \in \mathcal{D}$ with arbitrary perturbation vector ξ—representing data augmentation—and $\ell(z) = \log(1 + e^{-z})$ is the cross-entropy loss. We use the notation $L_{(t,\xi)}(\cdot) = L_{(x_t, y_t, \xi)}(\cdot)$ for convenience, and denote the 0-1 generalization error over the entire data stream $\mathcal{D}$ as $L_{\mathcal{D}}^{0-1}(\mathbf{W}) \triangleq \mathbb{E}_{(x,y) \sim \mathcal{D}}[\mathbb{1}\{y \cdot f_{\mathbf{W}}(x + \xi) < 0\}]$ with arbitrary perturbation vector ξ.

CSSL Formulation. Following Algorithm 5, our analysis considers a random initialization (`Initialize`)

[29]Assuming the widths of each hidden layer are the same is a commonly-used simplification. See [9, 29] for papers that have adopted the same assumption.

of model parameters $\mathbf{W}^{(0)} = \{W_1^{(0)}, \dots, W_L^{(0)}\}$ as shown below:

$$\begin{aligned} W_j^{(0)} &\sim \mathcal{N}\left(0, \frac{2}{m}\right), \; \forall\, j \in [L-1], \\ W_L^{(0)} &\sim \mathcal{N}\left(0, \frac{1}{m}\right). \end{aligned} \tag{20}$$

A buffer $\mathcal{R}$ of size C is used to maintain data for replay. All incoming data is added to $\mathcal{R}$ (`ReplayStore`), and an entry is randomly (uniformly) evicted from $\mathcal{R}$ (`ReplayEvict`) if $|\mathcal{R}| > C$.

Each online update takes B uniform samples from $\mathcal{R}$ (`ReplaySample`), forming a set $\mathcal{S}_t$ of replay examples at iteration t. The t-th update of model parameters $\mathbf{W}$ (`StreamingUpdate`) over new and replayed data is given by the following expression:

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \left(\nabla_{\mathbf{W}^{(t)}} L_{(x_t, y_t, \xi_t)}(\mathbf{W}^{(t)}) + \sum_{(x_{\text{rep}}, y_{\text{rep}}) \in \mathcal{S}_t} \nabla_{\mathbf{W}^{(t)}} L_{(x_{\text{rep}}, y_{\text{rep}}, \xi_{\text{rep}})}(\mathbf{W}^{(t)}) \right), \tag{21}$$

where η is the learning rate, ξ_t is an arbitrary perturbation vector for the t-th data stream example, and ξ_{rep} is an arbitrary perturbation vector that is separately generated for each individual replay example. Our theoretical analysis does not consider the compression of replay examples ($\texttt{Compress}(x) = x$).

Data Augmentation. The t-th example in the data stream $(x_t, y_t) \in \mathcal{D}$ is augmented (`Augment`) using the perturbation vector ξ_t; i.e., the network always observes an augmented version of the data, given by $(x_t + \xi_t, y_t)$. No assumptions are made about vectors ξ_t, though our final rate depends on their magnitude. Similarly, all replay samples are augmented via ξ_{rep}, such that the network observes $(x_{\text{rep}} + \xi_{\text{rep}}, y_{\text{rep}})$ for each $(x_{\text{rep}}, y_{\text{rep}}) \in \mathcal{S}_t$. All replay samples have unique perturbation vectors ξ_{rep}, but we use ξ_{rep} to denote all replay perturbation vectors for simplicity. We denote the set of all perturbation vectors used for new and replayed data throughout streaming as $\mathcal{X}$ and define $\Xi = \sup_{\xi \in \mathcal{X}} \|\xi\|_2^2$.

Neural Tangent Random Feature. For $\mathbf{W} \in \mathcal{W}$, we define the ω-neighborhood as follows:

$$\mathcal{B}(\mathbf{W}, \omega) \triangleq \{\mathbf{W}' \in \mathcal{W} : \|W_l' - W_l\|_F \leq \omega, \; \forall l \in [L]\}.$$

Given a set of all-zero weight matrices $\mathbf{0} \in \mathcal{W}$ and randomly initialized (according to (20)) $\mathbf{W}^{(0)}$, the Neural Tangent Random Feature (NTRF) [29] is then defined as the following function set

$$\mathcal{F}(\mathbf{W}^{(0)}, R) = \{f(\cdot) = f_{\mathbf{W}^{(0)}}(\cdot) + \langle \nabla_{\mathbf{W}} f_{\mathbf{W}^{(0)}}(\cdot), \mathbf{W} \rangle : \mathbf{W} \in \mathcal{B}(\mathbf{0}, R)\},$$

where $R > 0$ controls the size of the NTRF. We use the following shorthand to denote the first order Taylor approximation of network output, given weights $\mathbf{W}, \mathbf{W}' \in \mathcal{W}$ and input $x \in \mathbb{R}^d$:

$$F_{\mathbf{W}, \mathbf{W}'}(x) \triangleq f_{\mathbf{W}}(x) + \langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x), \mathbf{W}' - \mathbf{W} \rangle.$$

Using this shorthand, the NTRF can be alternatively formulated as:

$$\mathcal{F}(\mathbf{W}^{(0)}, R) = \{f(\cdot) = F_{\mathbf{W}^{(0)},\mathbf{W}'}(\cdot) : \mathbf{W}' - \mathbf{W}^{(0)} \in \mathcal{B}(\mathbf{0}, R)\}.$$

5.5.2 Convergence Guarantees

Here, we present the main result of our theoretical analysis for CSSL—an upper bound on the zero-one generalization error of networks trained via Algorithm 5, as described in Section 5.5.1. Proofs are deferred to Appendix D.5, but we provide a sketch in this section. Our analysis adopts the following set of assumptions.

Assumption 3. *For all data $(x_t, y_t) \in \mathcal{D}$, we assume $\|x_t\|_2 = 1$. Given two data examples $(x_i, y_i), (x_j, y_j) \in \mathcal{D}$, we assume $\|x_i - x_j\|_2 \geq \lambda$.*

The conditions in Assumption 3 are widely-used in analysis of overparameterized neural network generalization [9, 29, 151, 191, 275]. The unit norm assumption on the data is adopted for simplicity but can be relaxed to $c_1 \leq \|x_i\| \leq c_2$ for all $(x_i, y_i) \in \mathcal{D}$ and absolute constants $c_1, c_2 > 0$. Given these assumptions, we derive the following:

Theorem 6. *Assume Assumption 3 holds, $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$, and $m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$. Then, defining $\hat{\mathbf{W}}$ as a uniformly-random sample from iterates $\{\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)}\}$ obtained via Algorithm 5 with $\eta = \mathcal{O}\left(m^{-\frac{3}{2}}\right)$ and B replay samples chosen at every iteration from buffer $\mathcal{R}$, we have the following with probability at least $1-\delta$:*

$$\mathbb{E}\left[L_{\mathcal{D}}^{0-1}\left(\hat{\mathbf{W}}\right)\right] \leq \inf_{f \in \mathcal{F}(\mathbf{W}^{(0)}, R/m)} \left(\frac{4}{n}\sum_{t=1}^{n} \ell(y_t \cdot f(x_t + \xi_t))\right) + \sqrt{\frac{2\log(\frac{1}{\delta})}{n}} + \mathcal{O}\left(\frac{(1+\Xi)^{\frac{3}{4}}B^{\frac{1}{2}}n + R^2}{L^2\log^{\frac{3}{2}}(m)}\right)$$

where ξ_t is an arbitrary perturbation vector representing data augmentation applied at streaming iteration t, $R > 0$ is a constant controlling the size of the NTRF function class, and $\delta \in [0, 1)$.

Discussion. Theorem 6 provides an upper bound on the 0-1 generalization error over $\mathcal{D}$ for a network trained via Algorithm 5. This bound has three components. The first component—shaded in red—captures the 0-1 error achieved by the best function within the NTRF $\mathcal{F}(\mathbf{W}^{(0)}, R/m)$. Intuitively, this term captures the "classifiability" of the data—it has a large value when no function in the ω-neighborhood of $\mathbf{W}^{(0)}$ can fit the data well and vice versa. Given fixed m, however, this term can be made arbitrarily small by increasing R to enlarge the NTRF and, in turn, increase the size of the function space considered in the infimum.

The second term—shaded in green—is a probabilistic term that arises after invoking an online-to-batch conversion argument [31] to relate training error to 0-1 generalization error. This term contains $\log(1/\delta)$ in the numerator and n in the denominator. Smaller values of δ will make the bound looser but allow it to hold with higher probability. On the other hand, larger values of n make this term arbitrarily small, meaning that the bound can be improved by observing more data.

The final term—shaded in blue—considers all other additive terms that arise from technical portions of the proof. This asymptotic expression grows with Ξ, R, B, and n. However, the term $L^2\log^{\frac{3}{2}}(m)$ occurs in

Method	Base Init. Required	Ratio of Params Frozen	Uses Replay
ExStream [102]	✓	95%	✓
DeepSLDA [104]	✓	95%	✗
REMIND [103]	✓	75%	✓
CSSL	✗	0%	✓

Table 10: A basic outline of properties for each of the considered streaming algorithms.

the denominator, revealing that this final term will shrink as the model is made wider and deeper.[30] Thus, Theorem 6, as a whole, shows that the network generalizes well given sufficiently large m, L, and n—*meaning that the network is sufficiently overparameterized and enough data has been observed.*

Sketch. The proof of Theorem 6—provided in Appendix D.5—proceeds as follows:

- We first show that the average 0-1 loss of iterates $\{\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)}\}$ obtained during streaming can be upper bounded with a constant multiple of the average cross-entropy loss of the same iterates.
- Invoking Lemma 10, we convert this average loss of iterates $\{\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)}\}$ to an average loss of weights within the set $\mathbf{W}^\star \in \mathcal{B}(\mathbf{W}^{(0)}, R/m)$, where $R > 0$.
- Using an online-to-batch conversion argument (see Proposition 1 in [31]), we lower bound average 0-1 loss with expected 0-1 loss over $\mathcal{D}$ with probability $1 - \delta$.
- From here, we leverage Lemma 12 to relate average loss of $\mathbf{W}^\star$ over streaming iterations to the NTRF, taking an infimum over all functions $f \in \mathcal{F}(\mathbf{W}^{(0)}, R/m)$ to yield the final bound.

5.6 Experiments

Following prior work, we use ResNet18 to perform class-incremental streaming experiments on CIFAR100 and ImageNet [51, 144], experiments with various different data orderings on Core50 [164], and experiments using a novel, multi-task streaming learning setting. As baselines, we adopt the widely-used streaming algorithms ExStream [102], Deep SLDA [104], and REMIND [103]; see Table 10 for details. Because REMIND freezes most network layers after base initialization, we also study a REMIND variant ("Remind + Extra Params") with added, trainable layers such that the number of trainable parameters is equal to CSSL.[31]

All experiments split the dataset(s) into several batches and data ordering is fixed between comparable experiments. Evaluation occurs after each batch and baselines use the first batch for base initialization—the first batch is seen both during base initialization and streaming. CSSL performs no base initialization, but may begin the streaming process with pre-trained parameters—such experiments are identified accordingly. We

[30]Observe that n appears in the denominator of the green term and the numerator of the blue term of Theorem 6. However, because the model is assumed to be overparameterized (i.e., $m \gg n$), the blue term is still negligible even given values of n that are sufficiently large to make the green term small.

[31]This modified version of REMIND has the same number of trainable parameters as a normal ResNet18 model and is included as a baseline to ensure the benefit of CSSL is not simply due to the presence of more trainable parameters within the underlying network.

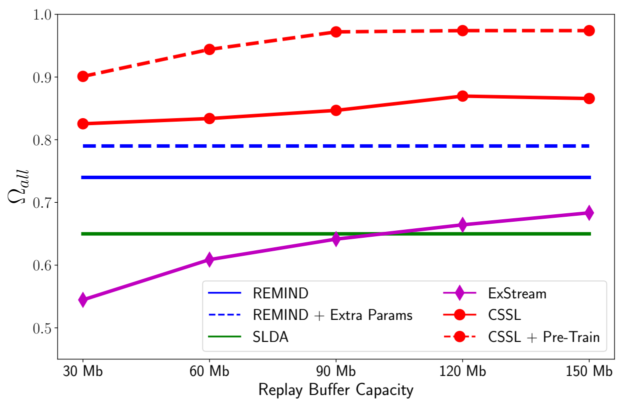

Figure 12: Streaming performance on CIFAR100 using different replay buffer capacities.

first present class-incremental streaming experiments, followed by the analysis of different data orderings and multi-task streaming. The section concludes with supplemental analysis that considers training other network architectures and studies behavioral properties of networks trained via CSSL.

5.6.1 Class-Incremental Learning

We perform class-incremental streaming learning experiments using a ResNet18 architecture on CIFAR100 and ImageNet. The results of these experiments are summarized below, and a discussion of the results follows.

CIFAR100. The dataset is divided into batches containing 20 classes and results are averaged over three different data orderings. Experiments are conducted using replay buffer memory capacities ranging from 30Mb (i.e., 10K CIFAR100 images) to 150Mb. These capacities are not relevant to Deep SLDA or REMIND, as Deep SLDA does not use a replay buffer and REMIND can store the full dataset using $<$ 30Mb of memory. CSSL compresses replay examples by quantizing pixels to 6 bits and resizing images to 66% of their original area. In some cases, CSSL is initialized with parameters that are pre-trained over the first 20 classes of CIFAR100; see Appendix D.1.1 for details. The results of these experiments are illustrated in Figure 12.

ImageNet. The dataset is divided into batches of 100 classes. Following [103], we perform experiments with replay buffer memory capacities of 1.5Gb and 8Gb. For reference, 1.5Gb and 8Gb experiments with (without) JPEG compression store 100,000 (10,000) and 500,000 (50,000) images for replay, respectively, given standard pre-processing. Such capacities are not relevant to Deep SLDA.

Several CSSL variants are tested that i) optionally pre-train over the base initialization set and ii) may store the replay buffer on disk. For in-memory replay, data is compressed by quantizing each pixel to four bits and resizing images to 75% of their original area. No quantization or resizing is employed when the replay buffer is stored on disk; see Appendix D.1.1 for details. ImageNet results are provided in Table 11.

Discussion. CSSL far outperforms baselines on CIFAR100. Even at the lowest buffer capacity with a random parameter initialization, our methodology exceeds the performance of REMIND by 3.6% absolute Ω_{all} given an equal number of trainable parameters, revealing that i) the performance improvement of CSSL is not solely due to having more trainable parameters and ii) the proposed approach is capable of achieving impressive

Method	Replay Buffer Size	
	1.5Gb	8Gb
ExStream	0.569	0.594
Deep SLDA	0.752	0.752
REMIND	0.855	0.856
REMIND + Extra Params	**0.869**	0.873
CSSL	0.740	0.873
CSSL + Pre-Train	0.750	**0.903**
CSSL + JPEG	0.899	0.951
CSSL + Pre-Train + JPEG	**0.909**	**0.964**

Table 11: Top-5 Ω_{all} on ImageNet with 1.5Gb and 8Gb buffer capacities. Methods that store the replay buffer on main memory and disk are separated by a horizontal line.

performance even with limited memory overhead. When a pre-trained parameter initialization is used, the performance improvement of CSSL is even more pronounced, reaching an absolute Ω_{all} of 0.974 that nearly matches offline performance. For comparison, REMIND achieves Ω_{all} of 0.790 with an equal number of trainable parameters and unlimited replay capacity.

On ImageNet, the proposed methodology without JPEG compression performs comparably to Deep SLDA at a memory capacity of 1.5Gb. Performance improves significantly—and surpasses baseline performance—as the replay buffer grows. For example, CSSL without JPEG compression matches or exceeds the performance of all baselines given a replay buffer capacity of 8Gb, and performance continues to improve when a pre-trained parameter initialization is used and as more images are retained for replay. In fact, using pre-trained parameters and 8Gb of memory on disk for replay, the proposed methodology can achieve a Top-5 Ω_{all} of 0.964, nearly matching offline training performance.

What's possible with more memory? REMIND is capable of storing the replay buffer with limited memory overhead and is the highest-performing method under strict memory limitations; e.g., REMIND achieves the best performance given a 1.5Gb buffer capacity on ImageNet assuming no access to disk storage. Despite the applicability of streaming learning to edge-device scenarios [102, 105], many streaming applications exist that do not induce strict memory requirements due to ease of access to high-end hardware on the cloud. Thus, one may begin to wonder whether better performance is possible when memory constraints upon the replay buffer are relaxed.

As outlined in Table 11, REMIND achieves an Ω_{all} of 0.856 (0.873 with equal trainable parameters to CSSL) when the entire dataset is stored for replay. Such performance is $> 9\%$ absolute Ω_{all} below the best-performing CSSL experiment (i.e., 8Gb capacity with JPEG storage and pre-training), which, despite increased memory overhead relative to REMIND, does not store the whole dataset in the replay buffer. The proposed approach continues improving as the replay buffer becomes larger, reaching a Top-5 Ω_{all} of 0.975 given unlimited replay capacity and a pre-trained parameter initialization. Although REMIND may be preferable in memory-constrained scenarios, *the proposed approach continues improving as more memory is allocated for replay, reaching performance levels much closer to that of offline training*.

Method	Top-1 Ω_{all} without Pre-Training				Top-1 Ω_{all} with Pre-Training			
	ID	CID	I	CI	ID	CID	I	CI
ExStream	0.719	0.635	0.734	0.627	0.959	0.936	0.954	0.935
Deep SLDA	0.721	0.660	0.730	0.626	0.971	0.952	0.962	0.957
REMIND	0.719	0.628	0.741	0.600	0.985	0.971	0.986	0.973
REMIND + Extra Params	0.636	0.586	0.693	0.594	**0.994**	0.961	**0.996**	0.966
CSSL (100Mb Buffer)	1.106	0.977	1.071	0.990	0.963	0.979	0.962	0.976
CSSL (200 Mb Buffer)	1.107	1.005	1.104	0.995	0.968	0.981	0.977	0.976
CSSL (300 Mb Buffer)	**1.153**	**1.009**	**1.112**	**1.024**	0.976	**0.986**	0.974	**0.979**

Table 12: Core50 streaming performance for different data orderings, averaged across ten permutations.

5.6.2 Different Data Orderings

We perform streaming experiments on the Core50 dataset using i.i.d. (ID), class i.i.d. (CID), instance (I), and class-instance (CI) orderings [164]. For each ordering, ten different permutations are generated, and performance is reported as an average across permutations. The dataset is split into batches of 1200 samples. Experiments are conducted with replay buffer memory capacities of 100Mb (i.e., 2,000 images from Core50, or $1/3$ of the full dataset), 200Mb, and 300Mb. These memory capacities only apply to CSSL, while baselines are given unlimited replay capacity. The proposed methodology compresses images using 4-bit integer quantization and resizing to 75% of original area; see Appendix D.1.2 for details.

Experiments are performed both with and without ImageNet pre-trained weights, allowing performance to be observed in scenarios without a high-quality parameter initialization. In experiments without ImageNet pre-training, baseline methodologies perform base initialization over the first 1200 dataset examples, while CSSL begins streaming from a random initialization. When ImageNet pre-training is used, CSSL begins the streaming process with these ImageNet pre-trained parameters, while baseline methodologies perform base initialization beginning from the pre-trained weights. *CSSL observes no data from the target domain prior to the commencement of streaming.* Core50 results are provided in Table 12.

Discussion. Without ImageNet pre-training, our methodology surpasses baseline performance at all memory capacities and often exceeds offline training performance. In this case, $\Omega_{\text{all}} > 1$ indicates that models trained via CSSL outperform offline-trained models during evaluation. This result is likely possible due to the use of simple augmentations (i.e., random crops and flips) for training offline models used to compute Ω_{all}; see Appendix D.1.2. Nonetheless, these findings highlight the broad applicability of CSSL. The proposed approach flourishes in this setting because it $i)$ has no dependence upon base initialization and $ii)$ is capable of learning all model parameters via streaming. In contrast, baseline methodologies struggle with the low-quality base initialization provided by only 1200 data examples.

When initialized with pre-trained weights, baseline methodologies perform much better, though the proposed methodology still matches or exceeds this performance. We emphasize that such pre-trained parameter initializations are not always possible; see Section 5.4. In numerous practical scenarios (e.g., surgical video, medical imaging, satellite or drone imaging, etc.), well-aligned, annotated public datasets may not exist. *The proposed methodology performs well with or without pre-training due to its end-to-end* *approach to streaming learning and even surpasses offline training performance when beginning streaming from a cold start.*

5.6.3 Multi-Task Streaming Learning

The Setting. In addition to our proposal of CSSL, we propose and explore a new, multi-task streaming learning domain. Although similar setups have been explored for batch incremental learning [118], no work has attempted multi-task learning in a streaming fashion, where multiple datasets with disjoint output spaces are learned sequentially, one example at a time. Such a setting is particularly interesting because the introduction of new datasets causes large shifts in properties of the data stream. In such scenarios, the learner must adapt to significant changes in incoming data, making the multi-task streaming learning domain an interesting test case for CSSL.

We consider multi-task streaming learning over several image classification datasets. In particular, we perform multi-task streaming learning over the CUB-200 [229], Oxford Flowers [188], MIT Scenes [197], and FGVC-Aircrafts datasets [172]. The datasets are learned in that order and are introduced to the data stream one example at a time in a class-incremental fashion; see Appendix D.1.3 for further details. Though the ordering of datasets is never changed, the ordering of data within each dataset may be changed. Comparable experiments use the same ordering of data. To handle the disjoint output spaces of each dataset, models for both the proposed methodology and baselines are modified to have a separate output layer for each dataset.

Methodology. For simplicity, all methodologies are given unlimited replay capacity for multi-task streaming learning experiments. Baseline methodologies perform base initialization over the first half of the MIT indoor dataset, optionally beginning with ImageNet pre-trained parameters. CSSL begins the streaming process with either random or ImageNet pre-trained parameters, such that no data from the target domain is observed prior to the commencement of streaming. All tests are performed over three distinct permutations of the data, and performance is averaged across permutations.

During streaming learning, each update performs $i)$ an update with replayed and new data for the dataset currently being learned and $ii)$ a separate, full update with replayed data for each of the datasets that have been learned previously. We find that such a strategy for multi-task replay performs better than other strategies, such as performing a single update that encompasses all datasets; see Appendix D.2.5. Deep SLDA and ExStream do not adopt this replay strategy, as only the final classification layers of the model are updated during streaming. Thus, multi-task streaming learning is identical to performing separate streaming experiments on each dataset. Performance in terms of Top-1 test accuracy is recorded separately over each dataset at the end of the streaming process, as shown in Table 13.[32]

Discussion As shown in Table 13, CSSL, both with and without pre-triaining, far outperforms all baseline methodologies on nearly all datasets in the multi-task streaming domain. For example, when ImageNet pre-training is used, CSSL outperforms REMIND by 10% absolute test accuracy (or more) on CUB-200, Oxford Flowers, and FGVC-Aircrafts datasets. When no ImageNet pre-training is used, the performance

[32]We choose to avoid reporting performance in terms of Ω_{all} and only report performance at the end of streaming to match the evaluation strategy of [118] and make results less difficult to interpret.

ImageNet Pre-Training	Method	Top-1 Accuracy			
		MIT Scenes	CUB-200	Flowers	FGVC
✓	Offline	71.42 ± 0.60	73.70 ± 0.01	91.98 ± 0.02	76.05 ± 0.17
	ExStream	58.96 ± 0.70	26.04 ± 0.46	69.16 ± 0.27	33.33 ± 0.56
	Deep SLDA	**60.05 ± 0.35**	29.85 ± 0.04	72.56 ± 0.13	34.16 ± 0.20
	REMIND	54.23 ± 0.58	46.27 ± 1.01	78.06 ± 1.47	36.56 ± 3.24
	CSSL	54.30 ± 1.95	**59.94 ± 0.97**	**88.36 ± 0.98**	**45.29 ± 2.11**
✗	Offline	51.60 ± 0.56	46.95 ± 0.25	83.10 ± 0.57	60.12 ± 0.44
	ExStream	31.84 ± 0.34	9.37 ± 0.08	40.31 ± 0.15	16.58 ± 0.20
	Deep SLDA	35.05 ± 0.19	11.30 ± 0.10	44.04 ± 0.09	16.89 ± 0.06
	REMIND	28.56 ± 0.71	17.97 ± 0.41	43.97 ± 1.29	16.53 ± 0.68
	CSSL	**42.49 ± 0.73**	**40.08 ± 0.45**	**78.70 ± 0.45**	**25.98 ± 2.30**

Table 13: Top-1 test accuracy of models trained with different methodologies for multi-task streaming learning.

improvement is even more drastic. In particular, CSSL provides a 22% and 35% absolute boost in Top-1 accuracy on the CUB-200 and Oxford Flowers datasets, respectively.

Interestingly, CSSL achieves lower performance than ExStream and Deep SLDA on the MIT Scenes dataset when ImageNet pre-training is used. The impressive performance of these baselines on MIT Scenes is caused by the fact that $i)$ ExStream and Deep SLDA only update the model's classification layer during streaming and $ii)$ the fixed feature extractor being used is pre-trained on ImageNet and fine-tuned on over half of the MIT Scenes dataset. Thus, ExStream and Deep SLDA perform well on this dataset only because their feature extractor has been extensively pre-trained and exposed to a large portion of the dataset during base initialization. When ImageNet pre-training is not included, the fixed feature extractor is of lower quality, leading ExStream and Deep SLDA to be outperformed by CSSL even on the MIT Scenes dataset.

Adapting the model's representations to new datasets is pivotal to achieving competitive performance in the multi-task streaming learning domain. Baseline methodologies—due to the fixing of parameters after base initialization—cannot adapt a majority of the model's parameters, leading their representations to be biased towards data observed during base initialization. Such a dilemma emphasizes the importance of developing streaming algorithms, such as CSSL, that learn end-to-end. By using CSSL for multi-task streaming learning, model representations are not fixed based on a subset of data that does not reflect future, incoming datasets, and model performance can benefit from the positive inductive transfer that occurs by updating all model parameters over multiple tasks or datasets simultaneously.

5.6.4 Supplemental Experiments and Analysis

We now present supplemental results that explore the properties of models trained with CSSL, as well as other model architectures.

Confidence Calibration Analysis. Confidence calibration characterizes the ability of a model to provide an accurate probability of correctness along with any of its predictions. In many cases, model softmax scores are

Dataset	Method	Average ECE
CIFAR100	ExStream	0.301 ± 0.008
	Deep SLDA	0.296 ± 0.012
	REMIND	**0.023 ± 0.000**
	CSSL + Pre-Train (30Mb Buffer)	0.031 ± 0.005
	CSSL + Pre-Train (150Mb Buffer)	**0.023 ± 0.003**
	CSSL + Pre-Train + No Aug. (150Mb Buffer)	0.161 ± 0.031
	CSSL (30Mb Buffer)	0.040 ± 0.001
	CSSL (150Mb Buffer)	0.036 ± 0.002
	CSSL + No Aug. (150Mb Buffer)	0.149 ± 0.015
ImageNet	REMIND	**0.021 ± 0.005**
	CSSL	0.043 ± 0.003
	CSSL + No Aug.	0.266 ± 0.009

Table 14: Average ECE of models trained via diferent streaming methodologies on class incremental streaming experiments with CIFAR100 and ImageNet datasets.

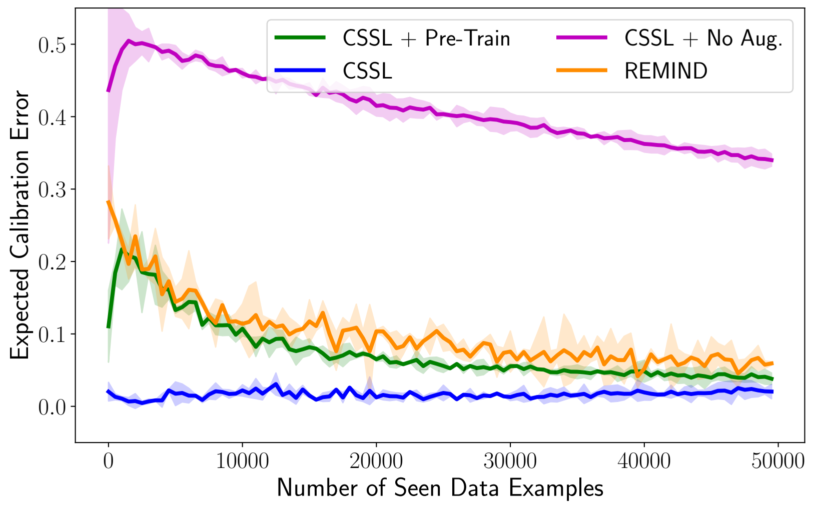

Figure 13: ECE of models trained via REMIND and CSSL on an i.i.d. ordering of CIFAR100.

erroneously interpreted as valid confidence estimates, though it is widely-known that deep networks make incorrect or poor predictions with high confidence [90].

Ideally, deep models should be highly-calibrated, as such a property would allow incorrect model predictions to be identified and discarded. For streaming learning, confidence calibration is especially useful, as one must decide during the streaming process whether the model's predictions should start to be relied upon. If the underlying model is highly-calibrated throughout streaming, this problem solves itself—just use predictions that are made with high-confidence, indicating a high probability of correctness.

Previous work indicates that using Mixup encourage good calibration properties [225], indicating that models obtained from CSSL and REMIND—both of which use some form of Mixup—should be highly-

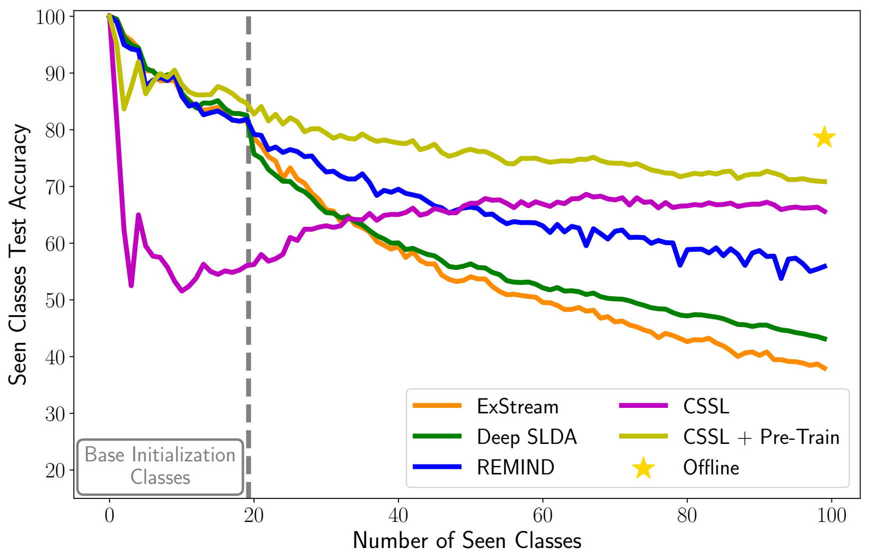

Figure 14: CIFAR100 class-incremental streaming performance measured after each new class is learned.

calibrated. In this section, we measure confidence calibration, in terms of average expected calibration error (ECE) across testing events, of models during class-incremental streaming experiments on CIFAR100 and ImageNet with different streaming methodologies; see Appendix D.1.4 for further details.

As shown in Table 14, CSSL and REMIND both aid model calibration. Interestingly, the confidence calibration of CSSL degrades significantly if only random crops and flips are used for augmentation (i.e., "CSSL + No Aug."), *highlighting the positive impact of a proper data augmentation policy on calibration.* Using an i.i.d. data ordering, we measure calibration more frequently throughout the streaming process in Figure 13 and find that CSSL is highly-calibrated throughout streaming when beginning from a random initialization and improves upon the calibration of REMIND when beginning from pre-trained parameters.

Performance Impact of a Cold Start. To better understand how model performance evolves throughout the streaming process, we perform class-incremental learning experiments on CIFAR100 and measure model accuracy after every new class that is introduced during streaming; see Appendix D.1.5 for more details.

As shown in Figure 14, baseline methodologies begin with a high accuracy but sharply decline in performance when the model encounters data not included in base initialization and continue to slowly degrade in performance throughout the remainder of the learning process. In contrast, randomly-initialized CSSL begins with relatively poor performance that improves gradually as more data is observed, eventually reaching a stable plateau in performance that surpasses all baseline methodologies. When beginning streaming from a pre-trained parameter initialization, this initial period of poor performance is eliminated, and the model still reaches a relatively stable plateau of higher performance.

Baseline methodologies perform well initially because of the data that was observed during base initialization, which is made evident by the sharp drop in baseline performance upon encountering new data; see Figure 14. Notably, CSSL does not experience this drop in accuracy even when beginning from pre-trained parameters, revealing that end-to-end training mitigates bias towards data used for base initialization. Furthermore, the ability of CSSL find a stable plateau in performance reveals that streaming learners need not always

Model	Top-5 Ω_{all}
ResNet101	0.872
MobileNetV2	0.904
DenseNet121	0.898
Wide ResNet50-2	0.880

Table 15: Class-incremental streaming performance with CSSL and various model architectures.

decay in performance as the data stream moves further away from data seen during base initialization. With CSSL, model representations can be adapted to changes in data over time to facilitate stable performance.

Different Network Architectures. Because CSSL trains the underlying network in an end-to-end fashion and is not dependent upon base initialization, its implementation is simple and agnostic to the particular choice of model architecture (i.e., substituting different architectures requires no implementation changes). Because a majority of prior work only studies ResNet architectures [102, 103], we use the proposed methodology to study the behavior of ResNet101 [106], MobileNetV2 [207], DenseNet121 [121], and Wide ResNet50-2 [260] architectures in class-incremental streaming experiments on ImageNet; see Appendix D.1.6 for further details. The replay buffer is stored on disk using JPEG compression with a memory capacity of 1.5Gb. The results of these experiments, recorded in terms of Top-5 Ω_{all}, are provided in Table 15, where it can be seen that CSSL achieves competitive performance with each of the different model architectures.

5.7 Conclusion

We present CSSL, a new approach to streaming learning with deep neural networks that trains the network end-to-end and uses a combination of replay mechanisms with sophisticated data augmentation to prevent catastrophic forgetting. Because the underlying network is learned end-to-end with CSSL, no base initialization is required prior to streaming, yielding a single-stage learning approach that is both simple and performant. In experiments, CSSL is found to surpass baseline performance—and even match or exceed offline training performance—on numerous established experimental setups, as well as on a multi-task streaming learning setup proposed in this work. We hope that CSSL inspires further developments in deep streaming learning by demonstrating the surprising effectiveness of simple learning techniques.

6 Better Schedules for Low Precision Training of Deep Neural Networks

Low precision training can significantly reduce the computational overhead of training deep neural networks (DNNs). Though many such techniques exist, cyclic precision training (CPT), which dynamically adjusts precision throughout training according to a cyclic schedule, achieves particularly impressive improvements in training efficiency, while actually improving DNN performance. Existing CPT implementations take common learning rate schedules (e.g., cyclical cosine schedules) and use them for low precision training without adequate comparisons to alternative scheduling options. We define a diverse suite of CPT schedules and analyze their performance across a variety of DNN training regimes, some of which are unexplored in the low precision training literature (e.g., node classification with graph neural networks). From these experiments, we discover alternative CPT schedules that offer further improvements in training efficiency and model performance, as well as derive a set of best practices for choosing CPT schedules. Going further, we find that a correlation exists between model performance and training cost, and that changing the underlying CPT schedule can control the tradeoff between these two variables. To explain the direct correlation between model performance and training cost, we draw a connection between quantized training and critical learning periods, suggesting that aggressive quantization is a form of learning impairment that can permanently damage model performance.

6.1 Introduction

DNNs have revolutionized the performance of autonomous systems. Yet, such gains come at a steep cost—DNN training is computationally burdensome and becoming more so, as the community tends towards larger-scale experiments [27]. Though many approaches exist for reducing DNN overhead [88, 148], low precision training has gained recent popularity due to its ability to improve training efficiency with minimal performance impact [75, 76].

Quantized training approaches use lower precision representations for DNN activations, gradients, and weights during training; see Figure 15. Then, the forward and backward pass can be expedited via (faster) low precision arithmetic, which recent generations of hardware are beginning to support [178]. Though implementing quantized training is more nuanced in complex architectures (e.g., batch normalization modules require special treatment [15]), the basic approach remains the same—DNN activations, weights, and gradients are replaced with lower-precision representations during training to reduce the cost of each forward and backward pass.

Early approaches to quantized training adopted a static approach that maintained the same, low level of precision throughout training [15, 249, 270]. Although this work was successful in revealing that DNN training is robust to substantial reductions in precision, later work achieved further efficiency gains by dynamically adapting precision along the training trajectory [75, 76]. Such work $i)$ sets minimum (q_{min}) and maximum (q_{max}) bounds for precision, and $ii)$ varies the precision between these bounds according to some (possibly cyclical) schedule throughout training. Interestingly, experiments using cyclic precision training (CPT) [75] demonstrated that precision plays a role similar to that of the learning rate in DNN training.

Because state-of-the-art results with DNNs are often achieved using curated hyperparameter schedules [53, 106, 214], different scheduling options for common hyperparameters (e.g., learning rate and momentum) have been explored extensively [37, 38]. Despite known similarities between precision and the learning rate, however, no such study has been performed for low precision training—existing approaches to CPT simply adopt common learning rate strategies (i.e., a cyclical cosine schedule [75]) without adequate comparison to alternatives. As such, *best practices for dynamically varying DNN precision along the training trajectory remain largely unknown*.

Our Proposal. To close this gap, we provide an extensive empirical study of different CPT variants. First, we rigorously define the space of potential CPT schedules by deconstructing such schedules via a three-step process of $i)$ choosing a profile, $ii)$ choosing the number of cycles, and $iii)$ choosing whether cycles are

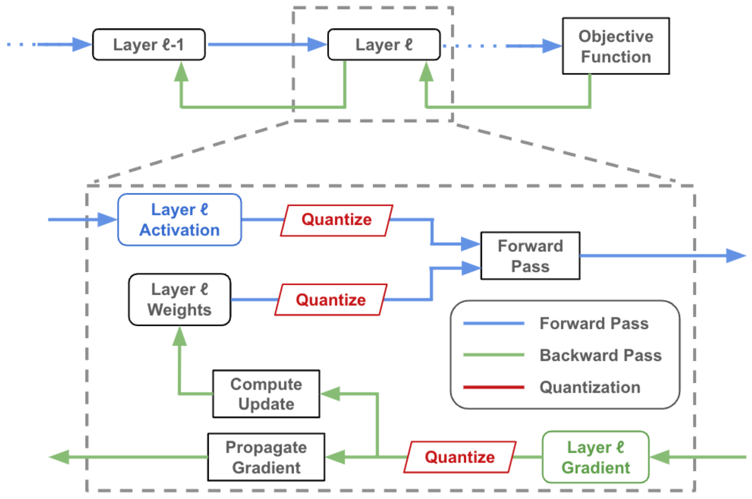

Figure 15: A depiction of quantized forward/backward pass within a single DNN layer.

repeated and/or have a "triangular" form. Using this decomposition, we define a suite of ten CPT schedules—including the originally-proposed cyclical cosine schedule [75]—that are analyzed across a variety of domains, including image classification and object detection with convolutional neural networks (CNNs), language modeling with long short-term memory (LSTM) networks [115], entailment classification with pre-trained transformers [53], and graph node classification with graph convolutional networks (GCNs) [141]. As a byproduct of this analysis, to the best of the authors' knowledge, we are the first to study quantized training strategies for GCNs.

Going beyond prior work, we discover new CPT variants that further improve DNN training efficiency and performance relative to prior schedules, revealing that exploring a wider variety of schedules for CPT is beneficial. More generally, we observe across all experiments that model performance tends to be correlated with the amount of compute used for training, revealing that manipulating the CPT schedule is a simple tool for controlling the tradeoff between model performance and training efficiency. Aiming to explain this relationship, we draw a connection between quantized training and critical learning periods, finding that prolonged training at low precision can impair the training process and cause a permanent performance degradation. Finally, we leverage these empirical observations to provide best practices for choosing the correct CPT schedule in different practical scenarios.

6.2 Related Work

Neural Network Quantization. Several works leverage DNN quantization to construct high-performing, efficient DNNs [25, 64, 192, 233, 248]. [126] studies quantization-aware training strategies to facilitate post-training DNN quantization, [132] adopts learnable approaches for DNN quantization, and [233] applies adaptive precision to different DNN components (e.g., layers or filters) during inference. Binary and ternary DNNs have also been studied [47, 147]. For GCNs, a few works have studied post-training quantization [69, 224], but quantized training of GCNs has not been considered.

Quantized Training. Several works, as in [15, 91, 178, 235], pioneer low precision DNN training, finding

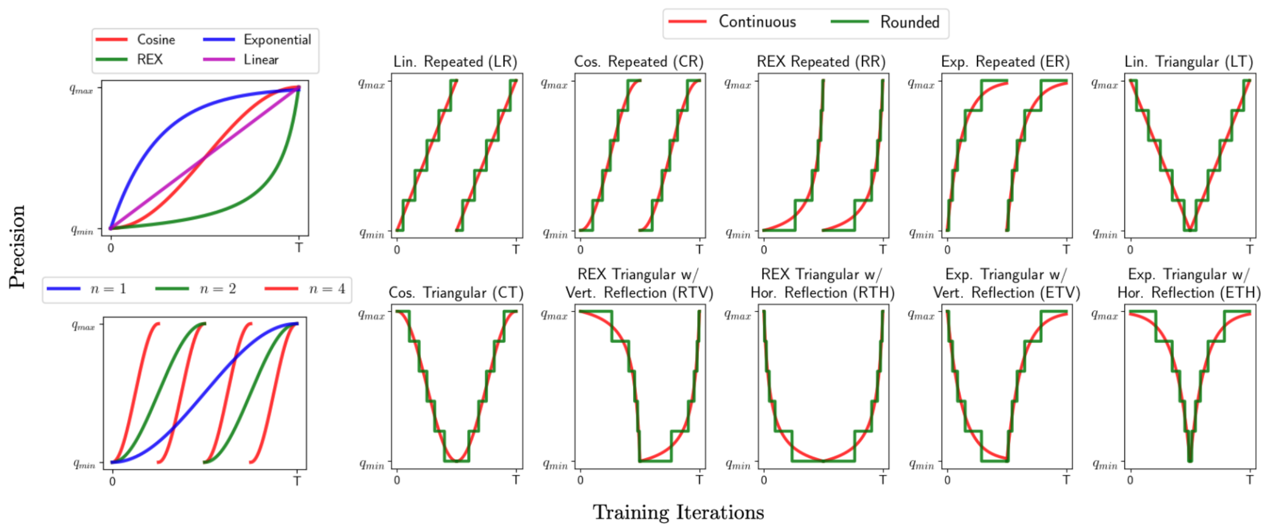

Figure 16: An illustration of profiles and schedules for CPT over T total training iterations. Function profiles are depicted in the upper-left subplot, while the lower-left subplot illustrates the CR schedule with different numbers of cycles n. Remaining subplots depict all possible CPT schedules explored in this work—both with and without rounding to the nearest integer—for $n = 2$ cycles.

that quantized training at a static, low level yields significant time and energy savings. Later work explores adaptive precision throughout training [75, 76]. Such methods can be applied on top of static quantization schemes and are found to yield gains in efficiency and model performance, by dynamically lowering precision below that of static techniques during training.

Critical Learning Periods. Critical learning periods within deep neural networks describe the early training epochs, during which the network is most sensitive to learning impairments. Early work on critical learning periods found that neural networks trained over blurry images for a sufficient number of epochs could never recover the accuracy of a network trained normally [3]. Subsequent work studied different forms of training impairments, such as improper regularization and non-i.i.d. data [11, 86]. Work on critical learning periods reveals that learning deficits during an initial, sensitive phase of training yield a permanent degradation in performance, no matter the amount of training performed after removing the deficit.

Hyperparameter Schedules. Most state-of-the-art results on deep learning benchmarks use curated hyperparameter schedules [106, 213]. Scheduling is commonly used for hyperparameters like the learning rate [166], but the general concept is widely-applied within deep learning. For example, scheduling approaches have been proposed for mini-batch sizes [244], image spatial resolution [68], the amount of regularization [215], and more. Best practices for hyperparameter scheduling have been established through extensive empirical analysis; e.g., for the learning rate [37, 213], momentum [38], and even batch size and weight decay [214]. Our work aims to provide empirical analysis that establishes such best practices for low precision training.

6.3 Precision Schedules for Quantized Training

Our goal in exploring different CPT schedules is to i) better understand the impact of such schedules on DNN performance, and ii) study new schedules that offer different levels of reductions in DNN training cost. We aim to make our analysis comprehensive by considering a wide variety of schedules. We will now provide a brief overview of CPT and explain how the suite of CPT schedules used in this work is derived.

6.3.1 How does CPT work?

`Quantize` in Figure 15 converts data into a lower precision representation, before it is used in the forward or backward pass. We refer to this lower level of precision as the "target" precision. CPT operates by simply varying this target precision such that each training iteration t has a different target precision q_t. More specifically, CPT varies target precision within the range $[q_{\text{min}}, q_{\text{max}}]$, according to some schedule during training. We define this schedule as a function $S(\cdot) : \mathbb{N}_{\geq 0} \rightarrow \mathcal{Q}$, where $\mathcal{Q}$ is the set of real-valued numbers in the range $[q_{\text{min}}, q_{\text{max}}]$. The precision used at iteration t of training, which is always rounded to the nearest integer, is given by $q_t = \texttt{round}(S(t)) \in [q_{\text{min}}, q_{\text{max}}]$.

q_{max} is selected to match the precision of static, low precision training, ensuring that CPT saves costs by adjusting DNN precision below this static level. q_{min} must be discovered via a precision range test (see Section 3.3 in [75]), as DNN training cannot progress when precision is too low. To stabilize training, cyclic precision is only applied to the forward pass (i.e., activation and weights quantization in Figure 15), while the backward pass (i.e., gradient quantization in Figure 15) fixes precision at q_{max} throughout training.

6.3.2 Constructing a CPT Schedule

Constructing a low precision training schedule can be decomposed into three steps:[33]

1. Selecting a Profile
2. Setting the Number of Cycles
3. Selecting Repeated or Triangular Cycles

The remainder of this section explains this decomposition and how it is used to derive the suite of CPT schedules explored in this work.

Step One: Selecting a Profile. Any CPT schedule must have an underlying function ("profile") according to which precision is varied. We consider four function profiles—cosine, linear, exponential, and reverse exponential (REX) [37]; see top left subplot of Figure 16. We only consider growth profiles (i.e., functions that increase precision from q_{min} to q_{max}), because DNN training must end with higher precision to facilitate convergence [75]. This set of functions characterizes a range of different quantization behaviors that reduce computational cost to varying degrees.

Step Two: Setting the Number of Cycles. Beyond choosing a profile, each schedule may perform a certain number of cycles, which we denote as n, according to this profile during training. Different choices for n are depicted in the bottom left subplot of Figure 16. In our experiments, we find that $n = 8$ performs consistently well.

[33]Related work [37] considers a sampling rate for each profile, which controls the frequency of hyperparameter updates. This sampling rate is less pertinent to precision schedules because precision is always rounded to the nearest integer, which makes updates less frequent.

Step Three: Selecting Repeated or Triangular Cycles. The proposed profiles can be used to produce many different schedules. Each cycle may repeatedly increase precision from q_{min} to q_{max}; see "repeated" subplots in Figure 16. In contrast, a schedule can be made "triangular" by reflecting the profile within every other cycle [75, 213, 214]; see top right and bottom repeated subplots in Figure 16. Assuming n is even and that all schedules end with a precision of q_{max} to facilitate model convergence, we can construct triangular schedules by reflecting all odd-numbered cycles, leading adjacent cycles to vary precision in opposite directions; see LT and CT subplots of Figure 16. REX and Exponential function profiles can be reflected either vertically or horizontally[34], producing two different triangular schedule variants; see the RTV, RTH, ETV and ETH subplots within Figure 16.

Suite of CPT Schedules. The full set of CPT schedules we consider is composed of all repeated and triangular variants of the linear, cosine, REX, and exponential function profiles. Notably, this set includes the original cyclical cosine schedule proposed for CPT (CR in Figure 16) [75]. Both repeated and triangular schedules can be repeated for any (even) number of cycles, though we set $n = 8$ in most experiments. The training efficiency of each schedule, relative to the others, does not change (assuming all schedules use the same setting of q_{min} and q_{max} for comparable experiments). With this in mind, we organize our suite of CPT schedules into three groups, which we refer to as Small, Medium, and Large based upon their provided reductions in training cost.

- Group I (Large): RR, RTH
- Group II (Medium): LR, LT, CR, CT, RTV, ETV
- Group III (Small): ER, ETH

To ease readability, we will use these groups to refer to different CPT schedules throughout the remainder of the text.

6.4 Experiments

We perform experiments across a variety of domains, including image recognition (i.e., image classification and object detection), node classification, and language understanding (i.e., language modeling and entailment classification). Results are evaluated based upon model performance and the computational cost of training. For each experimental domain, we first provide details about the setup, then present and analyze results. The section concludes with an empirically-derived set of practical suggestions for selecting a CPT schedule.

6.4.1 Preliminaries

We set $q_{\text{max}} \in \{6, 8\}$ and $n = 8$. q_{min} is derived for each model-dataset pair using a precision range test. Our baseline, inspired by SBM [15][35], maintains a static precision of q_{max} throughout training. To quantify the computational cost of training, we report the effective number of bit operations [270], which is proportional to the total number of bit operations during training. This quantity can be computed as:

$$\texttt{BitOps} = \texttt{FLOP}_{a\times b} \cdot \left({}^{\texttt{Bit}_a}\!/_{32}\right) \cdot \left({}^{\texttt{Bit}_b}\!/_{32}\right)$$

[34]Cosine and linear function profiles are symmetric, which causes horizontal and vertical reflections to be identical.

[35]Prior work [75] shows that SBM outperforms other static techniques for quantized training.

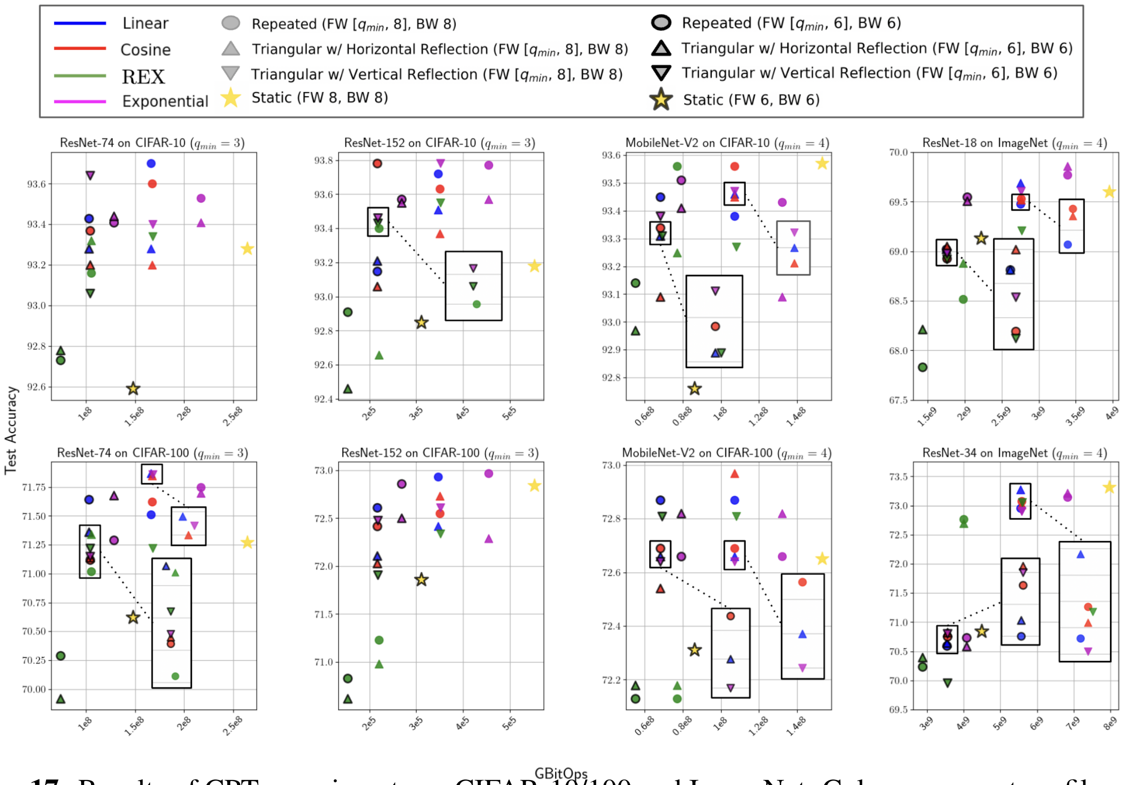

Figure 17: Results of CPT experiments on CIFAR-10/100 and ImageNet. Colors represent profiles, while shapes distinguish repeated or triangular schedules. Experiments are run with $q_{\max} \in \{6, 8\}$, distinguished by a dark outline around a shape. Future figures adopt the same scheme of colors and shapes.

for a dot product between a and b with precisions $\texttt{Bit}_a$ and $\texttt{Bit}_b$, respectively, and total number of floating point operations $\texttt{FLOP}_{a\times b}$.

Implementation. All image recognition experiments are implemented using PyTorch and TorchVision [193]. We implement GCN training using the Deep Graph Library [234], which provides a message passing framework for the communication of data within a graph, and PyTorch. For language modeling experiments, we base our implementation upon publicly available repositories for both language modeling and entailment classification [239, 261]. Experiments are run using a single Nvidia 3090 GPU. Because current generations of GPUs do not support arbitrary precision levels [178], we adopt the approach of prior work [75, 76] and simulate low precision training with different bit widths by clipping activation and gradient information beyond the current precision level q_t at every iteration of training.

6.4.2 Image Recognition

We evaluate the proposed CPT schedules on both image classification and object detection tasks. We train ResNet-74, ResNet-152, and MobileNet-V2 architectures on the CIFAR-10/100 datasets [106, 207], following the settings of [238]. Additionally, we adopt the settings of [106] and perform experiments with ResNet-18 and ResNet-34 architectures on ImageNet. Object detection experiments are performed on the PascalVOC dataset [67] and adopt a RetinaNet architecture [159] with an ImageNet pre-trained ResNet-18 backbone [106]. Performance is reported in terms of test accuracy and mean average precision (mAP) for image classification and object detection experiments, respectively. For all experiments, we report performance as an average across three separate trials and all hyperparameters are tuned using a hold-out validation set.

CIFAR-10/100. We adopt the settings of [238] for training. All models are trained for a total of $64K$ iterations with a batch size of 128 using standard crop and flip data augmentations. We use a SGDM optimizer with momentum of 0.9. Training begins with a learning rate of 0.1, and the learning rate is decayed by $10\times$ after 50% and 75% of training iterations. Weight decay is set to 1×10^{-4}, and we use $q_{\text{min}} = 3$ (i.e., discovered using a precision range test) and $q_{\text{max}} \in \{6, 8\}$. All CPT variants are tested with $n = 8$.

Results of experiments with CPT on CIFAR-10 and CIFAR-100 are shown in Figure 17. Despite using less training compute, CPT variants tend to outperform static baselines in nearly all cases. This finding aligns with prior work [75] but goes further by demonstrating that *the benefit of CPT is not specific to any particular precision schedule.* Compared to CPT with the originally-proposed CR schedule, our large schedules achieve further reductions in training compute, but slightly degrade accuracy. Small and medium schedules tend to both reduce training cost and improve accuracy.

Across all experiments, we see that a correlation exists between model performance and training compute, irrespective of model depth or architecture. This interesting finding reveals that *the choice of CPT schedule is a mechanism that controls the trade off between model performance and training compute*. Compared to most hyperparameter settings (e.g., the learning rate) that only impact model performance, the choice of CPT schedule is quite nuanced due to its joint impact on training efficiency and performance.

ImageNet. We consider the ILSVRC2012 version of ImageNet with 1K total classes. The settings of [106] are used: Models are trained for a total of 90 epochs with a batch size of 256 using standard crop and flip data augmentations. We use a SGDM optimizer with momentum of 0.9. An initial learning rate of 0.1 is adopted, and this learning rate is decayed by a factor of $10\times$ after 50% and 75% of total epochs. Weight decay is set to 1×10^{-5}, and we use $q_{\text{min}} = 4$ and $q_{\text{max}} \in \{6, 8\}$. All CPT variants are tested with $n = 8$.

As shown in Figure 17, CPT schedules that significantly reduce training compute cause a noticeable performance deterioration relative to static baselines on ImageNet. For example, large schedules cause a $0.5 - 1.5\%$ drop in accuracy relative to static baselines across different architectures and settings of q_{max}. With the larger ResNet-34 architecture, we see that more aggressive quantization is especially detrimental to performance; e.g., setting $q_{\text{max}} = 6$ significantly deteriorates performance in both CPT and baseline experiments.

Though we still observe a correlation between performance and training compute, larger-scale image classification experiments seem to be more sensitive to the level of training quantization—aggressive quantization noticeably deteriorates model performance. By adopting our small schedules that quantize more conservatively, however, we exceed baseline performance at a reduced training cost. In comparison, medium schedules—including the originally-proposed CR schedule—perform similarly to or worse than baseline models.

Pascal VOC. We consider the Pascal VOC 2012 dataset for both training and testing [67]. Prior to training on Pascal VOC, the ResNet backbone of each RetinaNet model is pre-trained on the ILSVRC2012 dataset. Models are trained for 120 total epochs with a batch size of 4 on Pascal VOC. We do not perform any data augmentation, though images are normalized and resized before being passed as input. We adopt an Adam

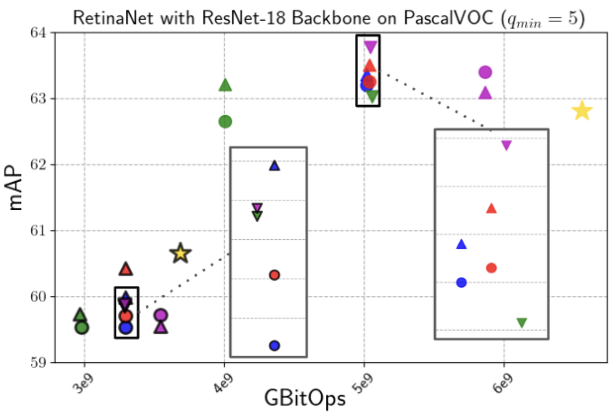

Figure 18: Results of CPT experiments on PascaVOC. The same coloring scheme is adopted from Figure 17.

optimizer [140] for training with a fixed learning rate of 1×10^{-5} throughout training. The model is trained using a focal loss that matches the settings of [159]. We use $q_{\text{min}} = 5$, $q_{\text{max}} \in \{6, 8\}$, and $n = 8$.

The results of CPT experiments on PascalVOC are shown in Figure 18. Here, we see that setting $q_{\text{max}} = 6$ yields a clear performance deterioration in both baseline and CPT experiments, indicating that training on PascalVOC is sensitive to quantization. When $q_{\text{max}} = 8$, however, *all CPT variants either match or exceed the performance of static baselinesm while reducing the cost of training*. For example, the best medium schedule (ETV) yeilds an absolute improvement of 1.02 mAP with a 25% reduction in training cost. The performance of all CPT schedules is roughly comparable when $q_{\text{max}} = 8$.

6.4.3 Node Classification

Graph node classification experiments are conducted with CPT on OGBN-Ariv and OGBN-Products datasets [120]. On OGBN-Arxiv, we use a regular GCN architecture [141] and consider the full graph during each training iteration. Experiments on OGBN-Products use a GraphSAGE architecture [92] with random neighbor sampling. Given that we are the first to explore low precision training within this domain, we first define low precision training with respect to the GCN architecture and identify the unique aspects of training GCNs with CPT.

Low Precision GCN Training. Consider a graph $\mathcal{G}$ comprised of e edges and n nodes with d-dimensional features $\mathbf{X} \in \mathbb{R}^{n \times d}$. The output $\mathbf{Y} \in \mathbb{R}^{n \times d_L}$ of a GCN is given by $\mathbf{Y} = \Psi_{\mathcal{G}}(\mathbf{X}; \mathbf{\Theta})$, where $\Psi_{\mathcal{G}}$ is a GCN architecture with L layers and (trainable) parameters $\mathbf{\Theta}$. Given $\mathbf{H}_0 = \mathbf{X}$, we have $\mathbf{Y} = \Psi_{\mathcal{G}}(\mathbf{X}; \mathbf{\Theta}) = \mathbf{H}_L$, with ℓ-th layer GCN representations

$$\mathbf{H}_\ell = \sigma(\bar{\mathbf{A}}\, \mathbf{H}_{\ell-1}\, \mathbf{\Theta}_{\ell-1}). \tag{22}$$

where $\sigma(\cdot)$ is an element-wise activation function (e.g., ReLU), $\bar{\mathbf{A}}$ is the degree-normalized adjacency matrix of $\mathcal{G}$ with added self-loops, and the trainable parameters $\mathbf{\Theta} = \{\mathbf{\Theta}_\ell\}_{\ell=0}^{L-1}$ have dimensions $\mathbf{\Theta}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ with $d_0 = d$ and $d_L = d'$. The GCN architecture is similar to a feed-forward neural network with an added node feature **aggregation step**, characterized by the left-multiplication of node features by $\bar{\mathbf{A}}$ in (22), at each

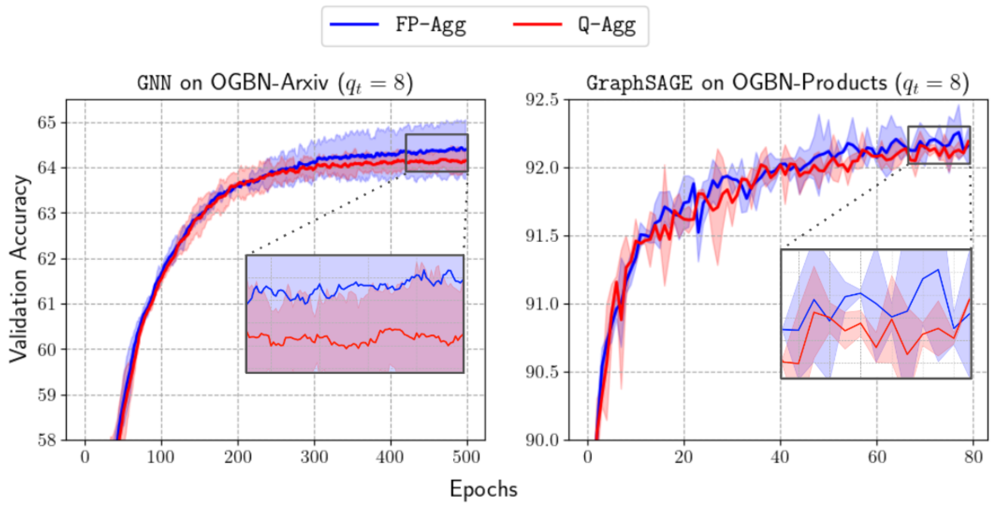

Figure 19: Validation accuracy of GCN and GraphSAGE models trained on OGBN-Arxiv and OGBN-Products using Q-Agg or FP-Agg and $q_{\text{max}} = q_t = 8$.

layer.

To apply quantized training GCNs, we must determine whether this aggregation step can be performed in low precision. We compare two strategies—FP-Agg and Q-Agg. Q-Agg indicates that the aggregation process is quantized—either according to a CPT schedule or a fixed precision level—within the GCN's forward pass. In contrast, FP-Agg always performs aggregation in full precision, no matter the precision level used in the rest of the network.

Node classification details. Before we present our findings, let us first provide the details of our experiments. For the OGBN-Arxiv dataset, we adopt a normal GCN model [141] with three layers and a hidden dimension of 512. Training on OGBN-Arxiv proceeds for 1000 epochs using an Adam optimizer [140]. The learning rate begins at an initial value of 10^{-3} and decays by a factor of $10\times$ over the course of training using cosine annealing. All CPT scheduling variants described in Section 6.3.2 are tested using $n = 8$. We set $q_{\text{min}} = 3$ (i.e., discovered using a precision range test [75]) and consider $q_{\text{max}} \in \{6, 8\}$.

For the OGBN-Products dataset, we use a two-layer GraphSAGE [92] model with a hidden dimension of 512. To make training computationally tractable over this large graph, we adopt random neighbor sampling with a neighborhood size of 32. We find that validation accuracy plateaus beyond a neighborhood of this size. Training proceeds for 80 epochs using an Adam optimizer [140]. The learning rate begins at an initial value of 3×10^{-4} and decays by a factor of $10\times$ over the course of training using cosine annealing. Again, we test all CPT scheduling variants with $n = 8$, $q_{\text{min}} = 3$ (i.e., discovered using a precision range test), and $q_{\text{max}} \in \{6, 8\}$.

Is aggregation robust to low precision? We perform experiments on OGBN-Arxiv and OGBN-Products to compare FP-Agg and Q-Agg strategies; see Figure 19. For these experiments, both strategies adopt a precision level of $q_t = q_{\text{max}} = 8$ throughout the training process.

On OGBN-Arxiv, utilizing the Q-Agg strategy yields a slight, but consistent, degradation in performance. The GCN trained using FP-Agg achieves a 0.5% absolute improvement in accuracy compared to a GCN

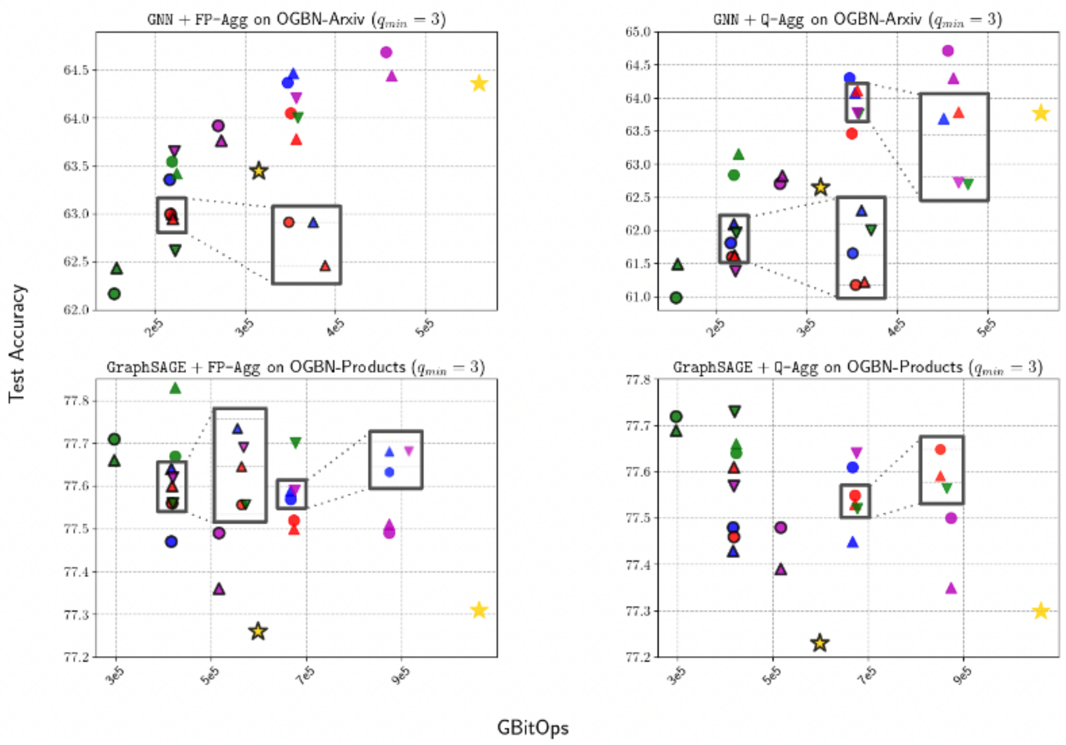

Figure 20: GCN and GraphSAGE test accuracy on OGBN-Arxiv and OGBN-Products. The coloring scheme is adopted from Figure 17.

trained with `Q-Agg`. This difference in performance is less pronounced on OGBN-products—`FP-Agg` and `Q-Agg` strategies train GraphSAGE models to similar accuracy.[36]

The aggregation process is a negligible portion of the GCN's forward pass. However, performing aggregation in low precision could greatly benefit communication efficiency in model-parallel training scenarios that pipeline the GCN's forward pass across multiple compute sites [231]. Though we do not consider this case in our work, we analyze both `FP-Agg` and `Q-Agg` strategies within the remainder of experiments due to this potential benefit of `Q-Agg` and its similar level of performance relative to `FP-Agg`.

OGBN-Arxiv. Results for the GCN model trained with different CPT schedules on OGBN-Arxiv are depicted in Figure 20. Similarly to experiments on image recognition, we observe a clear relationship between GCN test accuracy and the amount of compute used during training—schedules that perform aggressive quantization tend to be slightly outperformed by those quantizing more modestly. For example, for both `FP-Agg` and `Q-Agg`, models trained with CPT using large schedules reach an accuracy 1.0-1.5% below that of baseline models.

On the other hand, GCN's trained with medium schedules tend to match baseline performance in most cases, while models trained with small schedules outperform baselines in all cases. Node classification seems to be a complex task that is potentially sensitive to quantized training. However, by using alternative CPT

[36]The lesser impact of `Q-Agg` on OGBN-Products is due to the use of neighborhood sampling. The aggregation process computes a sum over all neighboring features, which can generate large, numerically unstable components unless the sum is truncated to a smaller, fixed number of neighboring features.

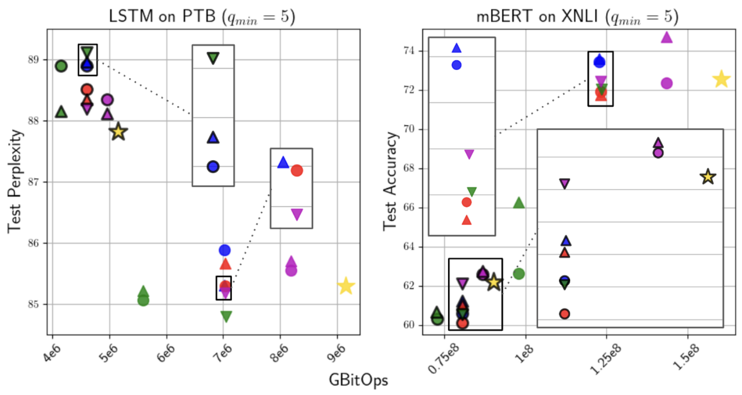

Figure 21: LSTM and mBERT results on Penn Treebank and the XNLI corpus, respectively. The coloring scheme is adopted from Figure 17.

schedules (i.e., medium or small schedules), we can achieve significant reductions in training compute, while actually improving the performance of the underlying model for both `FP-Agg` and `Q-Agg` CPT variants.

OGBN-Products Compared to experiments on OGBN-Arxiv, GraphSAGE training on OGBN-Products seems to be less sensitive to quantization; see Figure 20. Nearly all CPT schedules considered yield models that achieve better performance than the baseline models. For example, using both `FP-Agg` and `Q-Agg`, large schedules reduce the amount of training compute by $> 2\times$, while achieving a 0.3% to 0.5% absolute improvement in test accuracy relative to baselines. This improvement in test accuracy is standard across nearly all CPT schedules tested on OGBN-Products, indicating that this setting is relatively robust to low precision training. Significant reductions in training cost can be achieved—without damaging model performance—by using aggressive (large) quantization schedules with CPT.

6.4.4 Language Understanding

We perform language modeling experiments with CPT using LSTM networks on the Penn Treebank dataset. In these experiments, a one-layer LSTM model [115] is used to perform word-level language modeling following the settings of [261], and we evaluate performance in terms of perplexity. Additionally, we fine-tune a pre-trained, multilingual BERT (mBERT) [53] model (i.e., based on BERT-base-cased) with CPT on the Cross-lingual NLI (XNLI) corpus [46], where performance is measured in terms of test accuracy. In both settings, we report performance as an average across three trials and tune hyperparameters using grid search over a hold-out validation set. All results are illustrated within Figure 21.

Penn Treebank. Experiments follow the setup of [261]. We use a one-layer LSTM [115] model with a hidden dimension of 800. Dropout regularization with $p = 0.5$ is applied to the final output layer of the LSTM. Models are trained for 40 total epochs using a batch size of 20. All training occurs over sequences of length 35 that have been sampled from the Penn Treebank dataset. Training begins with a learning rate of 20, and the learning rate is divided by five each time the validation accuracy does not improve between epochs. Throughout training we clip gradients with a maximum norm of 0.25. For CPT, we adopt $q_{\text{min}} = 5$, $q_{\text{max}} \in \{6, 8\}$, and $n = 2$.

As shown in Figure 21, the language modeling setting is sensitive to quantization, as revealed by the clear degradation in model performance when $q_{\text{max}} = 6$. Similarly to experiments with PascalVOC in Section 6.4.2, however, all CPT variants perform well when $q_{\text{max}} = 8$. More specifically, we can reduce training cost from > 9 GBitOps to 5.5 GBitOps, while improving model perplexity by leveraging large CPT schedules. In comparison, the original CR schedule matches baseline perplexity at a cost of 7 GBitOps, again showing that *CPT performance can be improved by exploring alternative schedules within our proposed suite.*

XNLI. Experiments follow the settings of a fine-tuning example scripts within the HuggingFace transformers code repository [124, 239]. We use the BERT-base-cased-multilingual pre-trained model. Fine-tuning progresses for 2 total epochs with a batch size of 64 and a sequence length of 128. The initial learning rate is chosen from the set $\{5 \times 10^{-6}, 1 \times 10^{-5}, 5 \times 10^{-5}\}$ and is derived separately for each baseline model and CPT scheduling variant using a hold-out validation set. The learning rate is decayed linearly by $10\times$ throughout fine-tuning. For CPT, we use $q_{\text{min}} = 5$, $q_{\text{max}} \in \{6, 8\}$, and $n = 2$. Notable, we adopt a smaller value of n here because training on proceeds for 2 epochs, and we found that $n \in \{1, 2\}$ perform similarly.

In experiments with mBERT in Figure 21, we again see deteriorated performance when $q_{\text{max}} = 6$. When $q_{\text{max}} = 8$, we see a clear correlation between training compute and test accuracy. For example, large schedules, which require the fewest number of effective bit operations, perform significantly worse than other scheduling variants, while medium schedules tend to match baseline performance at a 25% reduction in compute costs. Most interestingly, large schedules (e.g., ETH) improve upon baseline performance by as much as 2% in absolute test accuracy and still reduce the total cost of training.

6.4.5 Best Practices for CPT

Practitioners will often be faced with choosing a single CPT schedule for training their model. There is no one CPT schedule that is “best” for every domain. Though the choice of CPT schedule is dependent upon several factors, we can provide the following insight for choosing the most appropriate schedule for a given application.

- **Minimizing training cost:** small schedules yield the largest efficiency gains (but may degrade model performance).
- **Maximizing model performance:** large schedules consistently match or exceed baseline accuracy, even in large-scale experiments.
- **Finding a balance:** medium schedules consistently reduce training cost while maintaining reasonable performance.

Such recommendations are reflective of the empirical correlation we observe between model performance and training cost in the majority of domains. In most cases, as expected, more aggressive quantization during training will come at the cost of slightly reduced model performance.

6.5 Connection to Critical Learning Periods

To better understand why we observe a direct relationship between model performance and total cost of training, we draw a connection between low precision training and critical learning periods in deep networks [3]. In particular, we show via experiments across multiple domains that low precision training is a form of training impairment that can cause permanent damage to network performance if applied too aggressively during the early, critical phase of learning.

6.5.1 Is low precision training a learning impairment?

We consider three settings—image classification with ResNet-74 on CIFAR-10, image classification with ResNet-18 on ImageNet, and node classification with GCNs on OGBN-Arxiv. All hyperparameter settings match those outlined in Sections 6.4.2 and 6.4.3.[37] Two types of experiments are performed:

- Train with low precision for R epochs or iterations at the beginning of training ($q_t = q_{\text{min}}$) for $t < R$), then use high precision for the remainder of training ($q_t = q_{\text{max}}$ for $t > R$).
- Train with low precision ($q_t = q_{\text{min}}$) during a span of training iterations and adopt a high precision level ($q_t = q_{\text{max}}$) outside of this span (i.e., “probing”).

The goal of these experiments is to determine $i)$ if low precision training impairs a neural network's learning process and $ii)$ whether this deficit or impairment is specific to the early, critical phase of learning.

Node Classification. Critical learning experiments with node classification on OGBN-Arxiv match the hyperparameter and architecture settings described above. In the first group of experiments, we begin by training GCNs with low precision for R epochs ($q_t = q_{\text{min}} = 3$ for $t < R$), where several different values of $R \in \{0, 100, 200, \dots, 1000\}$ are tested. After this initial period of low precision training, the GCN is further trained using higher precision ($q_t = q_{\text{max}} = 8$) for 1000 epochs (i.e., the normal training duration).

In the second group of experiments, we train the GCN for 2000 total epochs. During training, we perform low precision training ($q_t = q_{\text{min}} = 3$) for a total of 500 epochs. These 500 epochs of low precision training are placed at different points in the training process. In particular, we consider the following low precision training windows: $[0, 500]$, $[100, 600]$, $[200, 700]$, $[300, 800]$, $[400, 900]$.[38] Low precision training is performed in these windows, and normal precision ($q_t = q_{\text{max}} = 8$) is adopted outside of the windows. We test each window with a separate experiment.

The results of critical learning period experiments with OGBN-Arxiv are displayed in Figure 22. As shown in the leftmost subplot, the model's test accuracy deteriorates smoothly as the value of R increases, indicating that adopting low precision for a sufficient duration of time at the beginning of training can permanently damage model performance. The most significant deterioration in GCN test accuracy occurs when lower precision is maintained throughout the early training epochs, during which model accuracy improves the most

[37]The manner in which learning rate decay is performed could impact the final result of critical learning period experiments [3]. We test multiple learning rate decay strategies and find that they perform similarly. As such, we adopt a simple schedule that decays the learning rate normally throughout training.

[38]Here, each number represents an epoch. The window $[100, 600]$ means that low precision training was performed between epochs 100 and 600.

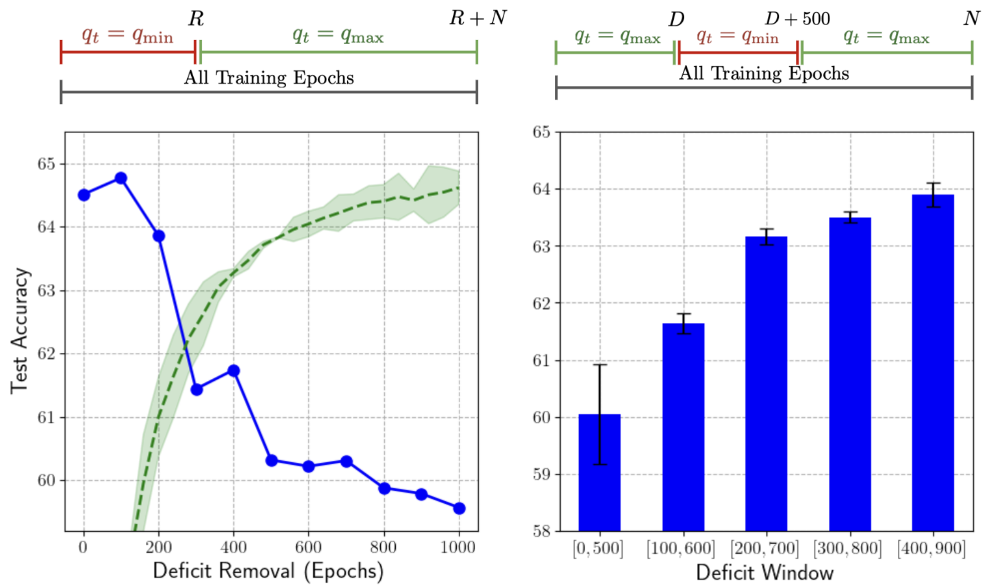

Figure 22: Test accuracy of GCNs trained on OGBN-Arxiv with different forms of learning impairments. (left-blue) Final accuracy of GCNs that are trained using $q_t = 3$ for R epochs, then trained normally with $q_t = 8$ for 1000 epochs. (left-green) Per-epoch test accuracy of a GCN trained normally with $q_t = 8$. (right) Final test accuracy of GCNs trained for a total 1000 epochs using $q_t = 8$, where a 500 epoch window using a lower precision of $q_t = 3$ is placed at different points within the training process.

(i.e., see the green curve in the left subplot of Figure 22). If this deficit is removed quickly, model performance does not deteriorate as drastically.

In probing experiments, shown in the right subplot of Figure 22, we see that applying window of low precision training yields the most noticeable accuracy deterioration at the beginning of the training process. Such a finding reveals that the performance deterioration associated with sufficiently-long periods of low precision training seems to be specific to the early, critical period of training. *Such a finding provides insight as to why CPT deteriorates model performance when aggressive quantization schedules are adopted.*

Image Classification. We perform similar critical learning period analysis in the image classification domain; see Table 16. On CIFAR-10, we again observe that test accuracy deteriorates smoothly as R increases, reaching a plateau around $R = 128K$. When a deficit window of $128K$ iterations is probed throughout the training process (i.e., bottom CIFAR-10 subsection in Table 16), we see that the impact of low precision training is most pronounced during the early training iterations. Due to computational expense, experiments on ImageNet only consider learning deficits applied at the beginning of training. Nonetheless, we again see that the test accuracy of models trained on ImageNet deteriorates as the value of R is increased. Even in large-scale experiments, network performance is sensitive to low precision training during the early part of training.

Discussion. Low precision training is a form of learning impairment that can cause permanent performance deterioration if applied during the early phase of learning. Such behavior is reminiscent of using improper regularization during training, which permanently impairs network performance if applied during early epochs [86]. Though training precision is known to impact the training process similarly to the learning

Setting	Deficit Window	Test Accuracy
ResNet-74 on CIFAR-10	None	92.23 ± 0.09
	[0, 16K]	92.03 ± 0.06
	[0, 32K]	92.01 ± 0.05
	[0, 64K]	92.04 ± 0.05
	[0, 128K]	91.64 ± 0.05
	[0, 256K]	91.25 ± 0.03
	[16K, 144K]	92.08 ± 0.20
	[32K, 160K]	92.16 ± 0.09
	[64K, 192K]	92.12 ± 0.04
ResNet-18 on ImageNet	None	67.44 ± 0.12
	[0, 25]	66.93 ± 0.06
	[0, 100]	66.65 ± 0.10

Table 16: Test accuracy of ResNets trained on CIFAR-10 and ImageNet with a low precision training deficit applied during different windows of time. Deficit windows are listed in terms of training iterations for CIFAR-10 and in terms of epochs for ImageNet.

rate [75], *it can also be viewed as a form of regularization*, which explains the correlation that is observed between training compute and model performance in Section 6.4. Although CPT can regularize the training process and improve model performance, schedules that apply quantization too aggressively during the critical period will experience a degradation in performance. As can be seen in Figure 22 and Table 16, *this problem can be solved by simply delaying the use of low precision until later during the training process.*

6.6 Conclusion

We perform an empirical analysis of different dynamic precision schedules for quantized training with DNNs. We find for low precision training techniques that a correlation exists between the amount of compute used during training and the model's performance, making the selection of a CPT schedule a simple tool for balancing performance and efficiency in DNN training. To explain this correlation, we draw a connection between low precision training and critical learning periods, finding that low precision training can permanently deteriorate model performance if applied during early epochs of training. We hope our findings will be helpful in furthering the adoption of quantized training techniques.

7 Closing Remarks

Despite the contributions of this work, adoption of deep learning-based technology is in its infancy. As emerging technologies continue to evolve, the most impactful techniques will continue to be those that focus upon performance, cost, and simplicity. We see this, for example, with the rising popularity of large language models [27], which leverage a simple, text-to-text interface for accurately solving a variety of different problems with a single model. Initially, such models were incredibly expensive to train, making them inaccessible to most practitioners [45, 217]. However, access was quickly democratized [226], such that large language models could be successfully trained and analyzed even within academic labs that lack industry-scale compute resources [42, 82]. Such accessibility has only further catalyzed the popularity and usage of large language models, making them one of the most widely-recognized deep learning-based technologies to date. Similarly, as existing and future topics in deep learning research continue to develop, including our own contributions, we argue that those ideas that provide simple, efficient, and accurate solutions to practical problems will have the most lasting impact.

As mentioned at the outset of this document, our research aims to make deep learning-based technology more practically useful. By focusing on modifications to the training process and the model itself, we have provided several proposals that tangibly improve the performance, cost, and complexity of deep learning. In Section 2, we propose a distributed training methodology for GCNs that can significantly accelerate large-scale experiments with modest compute resources. Through our exploration of provable pruning techniques in Sections 3 and 4, we enable state-of-the-art performance to be achieved with smaller DNNs and propose methods that significantly reduce the computational expense of DNN pruning. Our proposal of CSSL in Section 5 provides an intuitive and performant technique for adapting DNNs to incoming streams of data that is easy to implement, deploy, and analyze theoretically. Finally, the precision schedules analyzed in Section 6 show that simple hyperparameter scheduling techniques can improve DNN training efficiency and performance.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pages 265–283. Savannah, GA, USA, 2016.

[2] Manoj Acharya, Tyler L Hayes, and Christopher Kanan. Rodeo: Replay for online object detection. *arXiv preprint arXiv:2008.06439*, 2020.

[3] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

[4] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. *Advances in neural information processing systems*, 24, 2011.

[5] Rahaf Aljundi, Lucas Caccia, Eugene Belilovsky, Massimo Caccia, Min Lin, Laurent Charlin, and Tinne Tuytelaars. Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*, 2019.

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

[7] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*, 2019.

[8] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *arXiv preprint arXiv:1811.04918*, 2018.

[9] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

[10] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

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

[12] Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

[13] Bubacarr Bah and Jared Tanner. On the construction of sparse matrices from expander graphs. *Frontiers in Applied Mathematics and Statistics*, 4:39, 2018.

[14] Alexandru T Balaban. Applications of Graph Theory in Chemistry. *Journal of Chemical Information and Computer Sciences*, 1985.

[15] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018.

[16] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*, 2018.

[17] Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *arXiv preprint arXiv:1910.05422*, 2019.

[18] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *International conference on machine learning*, pages 583–593. PMLR, 2019.

[19] Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. A distributed frank-wolfe algorithm for communication-efficient sparse learning. In *Proceedings of the 2015 SIAM international conference on data mining*, pages 478–486. SIAM, 2015.

[20] Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 583–592, 2019.

[21] Eden Belouadah and Adrian Popescu. Scail: Classifier weights scaling for class incremental learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1266–1275, 2020.

[22] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. A comprehensive study of class incremental learning algorithms for visual tasks. *Neural Networks*, 2020.

[23] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

[24] Gil Benkö, Christoph Flamm, and Peter F Stadler. A graph-based toy model of chemistry. *Journal of Chemical Information and Computer Sciences*, 2003.

[25] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–697, 2020.

[26] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

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

[28] Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

[29] Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. *Advances in neural information processing systems*, 32, 2019.

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

[31] Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[32] Xiangyu Chang, Yingcong Li, Samet Oymak, and Christos Thrampoulidis. Provable benefits of overparameterization in model compression: From double descent to pruning neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6974–6983, 2021.

[33] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

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

[35] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

[36] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[37] John Chen, Cameron Wolfe, and Tasos Kyrillidis. Rex: Revisiting budgeted training with an improved schedule. *Proceedings of Machine Learning and Systems*, 4:64–76, 2022.

[38] John Chen, Cameron Wolfe, Zhao Li, and Anastasios Kyrillidis. Demon: Improved neural network training with momentum decay. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3958–3962. IEEE, 2022.

[39] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16306–16316, 2021.

[40] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

[41] Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020.

[42] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[43] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

[44] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Legr: Filter pruning via learned global ranking. 2019.

[45] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[46] Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018.

[47] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[48] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[49] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[50] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

[51] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[52] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[53] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[54] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5138–5146, 2019.

[55] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017.

[56] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

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

[58] Timothy J Draelos, Nadine E Miner, Christopher C Lamb, Jonathan A Cox, Craig M Vineyard, Kristofor D Carlson, William M Severa, Conrad D James, and James B Aimone. Neurogenesis deep learning: Extending deep networks to accommodate new classes. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 526–533. IEEE, 2017.

[59] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models

with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR, 2022.

[60] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685. PMLR, 2019.

[61] Chen Dun, Mirian Hipolito, Chris Jermaine, Dimitrios Dimitriadis, and Anastasios Kyrillidis. Efficient and light-weight federated learning via asynchronous distributed dropout. In *International Conference on Artificial Intelligence and Statistics*, pages 6630–6660. PMLR, 2023.

[62] Chen Dun, Cameron R Wolfe, Christopher M Jermaine, and Anastasios Kyrillidis. Resist: Layer-wise decomposition of resnets for distributed training. In *Uncertainty in Artificial Intelligence*, pages 610–620. PMLR, 2022.

[63] Alan Edelman and Yuyang Wang. Random matrix theory and its innovative applications. In *Advances in Applied Mathematics, Modeling, and Computational Science*, pages 91–116. Springer, 2013.

[64] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[65] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

[66] Utku Evci, Yani Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6577–6586, 2022.

[67] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[68] Fast.ai. Training a state-of-the-art model. `https://github.com/fastai/fastbook/blob/780b76bef3127ce5b64f8230fce60e915a7e0735/07_sizing_and_tta.ipynb`, 2020.

[69] Boyuan Feng, Yuke Wang, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding. Sgquant: Squeezing the last bit on graph neural networks with specialized quantization. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1044–1052. IEEE, 2020.

[70] Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[71] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[72] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[73] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

[74] Elias Frantar and Dan Alistarh. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

[75] Yonggan Fu, Han Guo, Meng Li, Xin Yang, Yining Ding, Vikas Chandra, and Yingyan Lin. Cpt: Efficient deep neural network training via cyclic precision. *arXiv preprint arXiv:2101.09868*, 2021.

[76] Yonggan Fu, Haoran You, Yang Zhao, Yue Wang, Chaojian Li, Kailash Gopalakrishnan, Zhangyang Wang, and Yingyan Lin. Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient dnn training. *Advances in Neural Information Processing Systems*, 33:12127–12139, 2020.

[77] Tommaso Furlanello, Jiaping Zhao, Andrew M Saxe, Laurent Itti, and Bosco S Tjan. Active long term memory networks. *arXiv preprint arXiv:1606.02355*, 2016.

[78] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[79] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[80] Jhair Gallardo, Tyler L Hayes, and Christopher Kanan. Self-supervised training enhances online continual learning. *arXiv preprint arXiv:2103.14010*, 2021.

[81] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1416–1424, 2018.

[82] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. Koala: A dialogue model for academic research. Blog post, April 2023.

[83] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. Integrated model, batch, and domain parallelism in training neural networks. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 77–86, 2018.

[84] Sharath Girish, Shishira R Maiya, Kamal Gupta, Hao Chen, Larry S Davis, and Abhinav Shrivastava. The lottery ticket hypothesis for object recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 762–771, 2021.

[85] Surbhi Goel, Adam Klivans, and Raghu Meka. Learning one convolutional layer with overlapping patches. In *International Conference on Machine Learning*, pages 1783–1791. PMLR, 2018.

[86] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. *Advances in Neural Information Processing Systems*, 32, 2019.

[87] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2005.

[88] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[89] Stefanie Gunther, Lars Ruthotto, Jacob B Schroder, Eric C Cyr, and Nicolas R Gauger. Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020.

[90] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[91] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015.

[92] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[93] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[94] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[95] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[96] Boris Hanin and Mihai Nica. Finite depth and width corrections to the neural tangent kernel. *arXiv preprint arXiv:1909.05989*, 2019.

[97] Karen Hao. Training a single ai model can emit as much carbon as five cars in their lifetimes (2019). *URL https://www. technologyreview. com/2019/06/06/239031*.

[98] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision*, pages 1–16. Springer, 2020.

[99] Tyler L. Hayes. Exstream. `https://github.com/tyler-hayes/ExStream`, 2019.

[100] Tyler L. Hayes. Deep slda. `https://github.com/tyler-hayes/Deep_SLDA`, 2020.

[101] Tyler L. Hayes. Remind. `https://github.com/tyler-hayes/REMIND`, 2020.

[102] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776. IEEE, 2019.

[103] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pages 466–483. Springer, 2020.

[104] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 220–221, 2020.

[105] Tyler L Hayes and Christopher Kanan. Online continual learning for embedded devices. *arXiv preprint arXiv:2203.10681*, 2022.

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

[107] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[108] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[109] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.

[110] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[111] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[112] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[113] Chris Hettinger, Tanner Christensen, Ben Ehlert, Jeffrey Humpherys, Tyler Jarvis, and Sean Wade. Forward thinking: Building and training neural networks one layer at a time. *arXiv preprint arXiv:1706.02480*, 2017.

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

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

[116] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[117] Jie Hou, Xianlin Zeng, Gang Wang, Jian Sun, and Jie Chen. Distributed momentum-based frank-wolfe algorithm for stochastic optimization. *IEEE/CAA Journal of Automatica Sinica*, 2022.

[118] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Lifelong learning via progressive distillation and retrospection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 437–452, 2018.

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

[120] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[121] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[122] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.

[123] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[124] HuggingFace. Text classification examples. `https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification`, 2023.

[125] Hiroshi Inoue. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929*, 2018.

[126] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[127] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[128] Gauri Jagatap and Chinmay Hegde. Learning relu networks via alternating minimization. *arXiv preprint arXiv:1806.07863*, 2018.

[129] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International conference on machine learning*, pages 427–435. PMLR, 2013.

[130] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

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

[132] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[133] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. 2020.

[134] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998.

[135] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 1998.

[136] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[137] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[138] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[139] Rajiv Khanna and Anastasios Kyrillidis. Iht dies hard: Provable accelerated iterative hard thresholding. In *International Conference on Artificial Intelligence and Statistics*, pages 188–198. PMLR, 2018.

[140] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[141] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[142] Andrew Kirby, Siddharth Samsi, Michael Jones, Albert Reuther, Jeremy Kepner, and Vijay Gadepally. Layer-parallel training with gpu concurrency of deep residual neural networks via nonlinear multigrid. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2020.

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

[144] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

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

[146] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[147] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[148] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[149] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[150] Xingguo Li, Junwei Lu, Zhaoran Wang, Jarvis Haupt, and Tuo Zhao. On tighter generalization bound for deep neural networks: Cnns, resnets, and beyond. *arXiv preprint arXiv:1806.05159*, 2018.

[151] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. *Advances in Neural Information Processing Systems*, 31, 2018.

[152] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. *Advances in neural information processing systems*, 30, 2017.

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

[154] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems*, 30, 2017.

[155] Fangshuo Liao and Anastasios Kyrillidis. On the convergence of shallow neural network training with randomly masked neurons. *arXiv preprint arXiv:2112.02668*, 2021.

[156] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*, 2019.

[157] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. *Advances in Neural Information Processing Systems*, 32:6665–6675, 2019.

[158] Tao Lin, Sebastian U Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

[159] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[160] Kuang Liu. Pytorch-cifar. `https://github.com/kuangliu/pytorch-cifar`, 2017.

[161] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019.

[162] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[163] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[164] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26. PMLR, 2017.

[165] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30:6467–6476, 2017.

[166] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[167] Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 12-in-1: Multi-task vision and language representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10437–10446, 2020.

[168] Yiping Lu, Chao Ma, Yulong Lu, Jianfeng Lu, and Lexing Ying. A mean field analysis of deep resnet and beyond: Towards provably optimization via overparameterization from depth. In *International Conference on Machine Learning*, pages 6426–6436. PMLR, 2020.

[169] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. Estimating the carbon footprint of bloom, a 176b parameter language model. *arXiv preprint arXiv:2211.02001*, 2022.

[170] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[171] Dean Lusher, Johan Koskinen, and Garry Robins. *Exponential random graph models for social networks: Theory, methods, and applications*. Cambridge University Press, 2013.

[172] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[173] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR, 2020.

[174] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2015.

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

[176] Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit. In *Conference on Learning Theory*, pages 2388–2464. PMLR, 2019.

[177] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[178] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[179] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[180] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *Advances in neural information processing systems*, 32, 2019.

[181] Ben Mussay, Margarita Osadchy, Vladimir Braverman, Samson Zhou, and Dan Feldman. Data-independent neural pruning via coresets. *arXiv preprint arXiv:1907.04018*, 2019.

[182] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[183] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[184] Deanna Needell and Joel A Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and computational harmonic analysis*, 26(3):301–321, 2009.

[185] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.

[186] Mark EJ Newman, Duncan J Watts, and Steven H Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 2002.

[187] Duy-Kien Nguyen and Takayuki Okatani. Multi-task learning of hierarchical vision-language representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10492–10501, 2019.

[188] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[189] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In *International Conference on Machine Learning*, pages 4839–4850. PMLR, 2019.

[190] Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic pruning is all you need. *Advances in Neural Information Processing Systems*, 33:2925–2934, 2020.

[191] Samet Oymak and Mahdi Soltanolkotabi. Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks. *IEEE Journal on Selected Areas in Information Theory*, 1(1):84–105, 2020.

[192] Eunhyeok Park and Sungjoo Yoo. Profit: A novel training method for sub-4-bit mobilenet models. In *European Conference on Computer Vision*, pages 430–446. Springer, 2020.

[193] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[194] J Gregory Pauloski, Zhao Zhang, Lei Huang, Weijia Xu, and Ian T Foster. Convolutional neural network training with distributed k-fac. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2020.

[195] Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. *Advances in neural information processing systems*, 33:2599–2610, 2020.

[196] Sebastian Pokutta, Christoph Spiegel, and Max Zimmer. Deep neural network training with frank-wolfe. *arXiv preprint arXiv:2010.07243*, 2020.

[197] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE conference on computer vision and pattern recognition*, pages 413–420. IEEE, 2009.

[198] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

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

[200] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020.

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

[202] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[203] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[204] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

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

[206] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[207] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[208] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29:93–93, 2008.

[209] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

[210] Shaohuai Shi, Zhenheng Tang, Xiaowen Chu, Chengjian Liu, Wei Wang, and Bo Li. A quantitative survey of communication optimizations in distributed deep learning. *IEEE Network*, 35(3):230–237, 2020.

[211] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*, 2017.

[212] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[213] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.

[214] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[215] Leslie N Smith. General cyclical training of neural networks. *arXiv preprint arXiv:2202.08835*, 2022.

[216] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pages 369–386. SPIE, 2019.

[217] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[218] Chaehwan Song, Ali Ramezani-Kebrya, Thomas Pethick, Armin Eftekhari, and Volkan Cevher. Subquadratic overparameterization for shallow neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[219] Mei Song, Andrea Montanari, and P Nguyen. A mean field view of the landscape of two-layers neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[220] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *jmlr*, 2014.

[221] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[222] Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR, 2019.

[223] Xavier Suau, Nicholas Apostoloff, et al. Filter distillation for network compression. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 3129–3138. IEEE, 2020.

[224] Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000*, 2020.

[225] Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[226] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[227] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[228] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447. PMLR, 2019.

[229] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[230] Hoi-To Wai, Jean Lafond, Anna Scaglione, and Eric Moulines. Decentralized frank–wolfe algorithm for convex and nonconvex problems. *IEEE Transactions on Automatic Control*, 62(11):5522–5537, 2017.

[231] Cheng Wan, Youjie Li, Cameron R Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. Pipegcn: Efficient full-graph training of graph convolutional networks with pipelined feature communication. *arXiv preprint arXiv:2203.10428*, 2022.

[232] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[233] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[234] Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[235] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31, 2018.

[236] Qihan Wang, Chen Dun, Fangshuo Liao, Chris Jermaine, and Anastasios Kyrillidis. Loft: Finding lottery tickets through filter-wise training. In *International Conference on Artificial Intelligence and Statistics*, pages 6498–6526. PMLR, 2023.

[237] Shuo Wang, Leandro L Minku, and Xin Yao. A systematic study of online class imbalance learning with concept drift. *IEEE transactions on neural networks and learning systems*, 29(10):4802–4821, 2018.

[238] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[239] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[240] Cameron R Wolfe and Keld T Lundgaard. E-stitchup: Data augmentation for pre-trained embeddings. *arXiv preprint arXiv:1912.00772*, 2019.

[241] Cameron R Wolfe and Keld T Lundgaard. Exceeding the limits of visual-linguistic multi-task learning. *arXiv preprint arXiv:2107.13054*, 2021.

[242] Joseph Worsham and Jugal Kalita. Multi-task learning for natural language processing in the 2020s: where are we going? *Pattern Recognition Letters*, 136:120–126, 2020.

[243] Chao-Yuan Wu, Ross Girshick, Kaiming He, Christoph Feichtenhofer, and Philipp Krahenbuhl. A multigrid method for efficiently training video models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[244] Chao-Yuan Wu, Ross Girshick, Kaiming He, Christoph Feichtenhofer, and Philipp Krahenbuhl. A multigrid method for efficiently training video models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 153–162, 2020.

[245] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019.

[246] Wenhan Xian, Feihu Huang, and Heng Huang. Communication-efficient frank-wolfe algorithm for nonconvex decentralized distributed learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10405–10413, 2021.

[247] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[248] Yuhui Xu, Shuai Zhang, Yingyong Qi, Jiaxian Guo, Weiyao Lin, and Hongkai Xiong. Dnq: Dynamic network quantization. *arXiv preprint arXiv:1812.02375*, 2018.

[249] Yukuan Yang, Lei Deng, Shuang Wu, Tianyi Yan, Yuan Xie, and Guoqi Li. Training high-performance and large-scale deep neural networks with full 8-bit integers. *Neural Networks*, 125:70–82, 2020.

[250] Mao Ye. Network-pruning-greedy-forward-selection. `https://github.com/lushleaf/Network-Pruning-Greedy-Forward-Selection`, 2021.

[251] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*, pages 10820–10830. PMLR, 2020.

[252] Mao Ye, Lemeng Wu, and Qiang Liu. Greedy optimization provably wins the lottery: Logarithmic number of winning tickets is enough. *Advances in Neural Information Processing Systems*, 33:16409–16420, 2020.

[253] Shihui Yin, Kyu-Hyoun Kim, Jinwook Oh, Naigang Wang, Mauricio Serrano, Jae-Sun Seo, and Jungwook Choi. The sooner the better: Investigating structure of early winning lottery tickets. 2019.

[254] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

[255] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. L2-gcn: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2127–2135, 2020.

[256] Kwangmin Yu, Thomas Flynn, Shinjae Yoo, and Nicholas D'Imperio. Layered sgd: A decentralized and synchronous sgd algorithm for scalable deep neural network training. *arXiv preprint arXiv:1906.05936*, 2019.

[257] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9194–9203, 2018.

[258] Binhang Yuan, Cameron R Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. Distributed learning of fully connected neural networks using independent subnet training. *Proceedings of the VLDB Endowment*, 15(8):1581–1590, 2022.

[259] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

[260] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[261] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[262] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[263] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[264] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[265] Shuai Zhang, Meng Wang, Sijia Liu, Pin-Yu Chen, and Jinjun Xiong. Why lottery ticket wins? a theoretical perspective of sample complexity on pruned neural networks. *arXiv preprint arXiv:2110.05667*, 2021.

[266] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. *Advances in neural information processing systems*, 28, 2015.

[267] Xiao Zhang, Yaodong Yu, Lingxiao Wang, and Quanquan Gu. Learning one-hidden-layer relu networks via gradient descent. In *The 22nd international conference on artificial intelligence and statistics*, pages 1524–1534. PMLR, 2019.

[268] Zhaoning Zhang, Lujia Yin, Yuxing Peng, and Dongsheng Li. A quick survey on large scale distributed deep learning systems. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1052–1056. IEEE, 2018.

[269] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

[270] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[271] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[272] Wentao Zhu, Can Zhao, Wenqi Li, Holger Roth, Ziyue Xu, and Daguang Xu. Lamp: Large deep nets with automated model parallelism for image segmentation. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part IV 23*, pages 374–384. Springer, 2020.

[273] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *Advances in neural information processing systems*, 31, 2018.

[274] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010.

[275] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep relu networks. *arXiv preprint arXiv:1811.08888*, 2018.

[276] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 32, 2019.

A GIST: Distributed Training for Large-Scale Graph Convolutional Networks

A.1 Experimental Details

A.1.1 Datasets

The details of the datasets utilized within GIST experiments in Section 2.4 are provided in Table 17. Cora, Citeseer, PubMed and OGBN-Arxiv are considered "small-scale" datasets and are utilized within experiments in Section 2.4.1. Reddit and Amazon2M are considered "large-scale" datasets and are utilized within experiments in Section 2.4.2.

Dataset	n	# Edges	# Labels	d
Cora	2 708	5 429	7	1,433
CiteSeer	3 312	4 723	6	3,703
Pubmed	19 717	44 338	3	500
OGBN-Arxiv	169 343	1.2M	40	128
Reddit	232 965	11.6 M	41	602
Amazon2M	2.5 M	61.8 M	47	100

Table 17: Details of relevant datasets.

A.1.2 Implementation Details

We provide an implementation of GIST in PyTorch [193] using the NCCL distributed communication package for training GCN [141], GraphSAGE [92] and GAT [227] architectures. Our implementation is centralized, meaning that a single process serves as a central parameter server. From this central process, the weights of the global model are maintained and partitioned to different worker processes (including itself) for independent training. Experiments are conducted with 8 NVIDIA Tesla V100-PCIE-32G GPUs, a 56-core Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, and 256 GB of RAM.

A.1.3 Small-Scale Experiments

Small-scale experiments in Section 2.4.1 are performed using Cora, Citeseer, Pubmed, and OGBN-Arxiv datasets [120, 208]. GIST experiments are performed with two, four, and eight sub-GCNs in all cases. We find that the performance of models trained with GIST is relatively robust to the number of local iterations ζ, but test accuracy decreases slightly as ζ increases; see Figure 24. Based on the results in Figure 24, we adopt $\zeta = 20$ for Cora, Citeseer, and Pubmed, as well as $\zeta = 100$ for OGBN-Arxiv.

Experiments are run for 400 epochs with a step learning rate schedule (i.e., $10\times$ decay at 50% and 75% of total epochs). A vanilla GCN model, as described in [141], is used. The model is trained in a full-batch manner using the Adam optimizer [140]. No node sampling techniques are employed because the graph is small enough to fit into memory. All reported results are averaged across five trials with different random

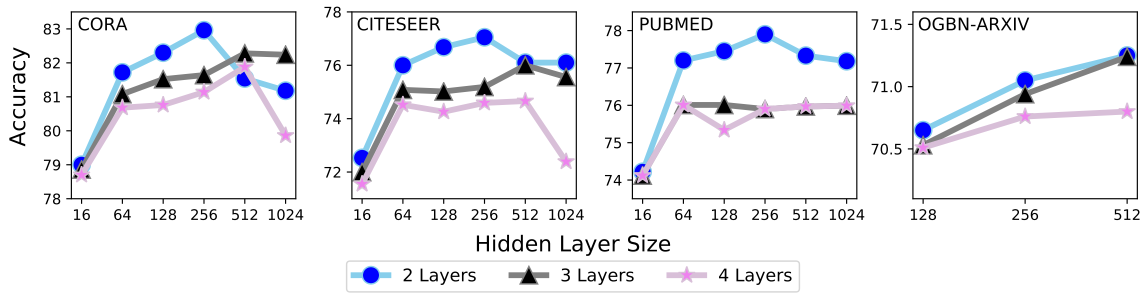

Figure 23: Test accuracy for different sizes (i.e., varying depth and width) of GCN models trained with standard, single-GPU methodology on small-scale datasets. We adopt three-layer, 256-dimensional GCN models as our baseline architecture.

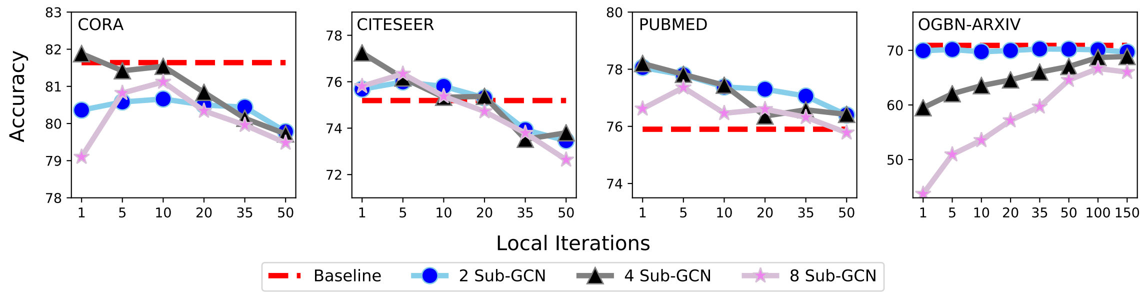

Figure 24: Test accuracy of GCN models trained on small-scale datasets with GIST using different numbers of local iterations and sub-GCNs.

seeds. For all models, d_0 and d_L are respectively given by the number of features and output classes in the dataset. The size of all hidden layers is the same, but may vary across experiments.

We first train baseline GCN models of different depths and hidden dimensions using a single GPU to determine the best model depth and hidden dimension to be used in small-scale experiments. The results are shown in Figure 23. Deeper models do not yield performance improvements for small-scale datasets, but test accuracy improves as the model becomes wider. Based upon the results in Figure 23, we adopt a three-layer GCN with a hidden dimension of $d_1 = d_2 = 256$ as the underlying model used in small-scale experiments. Though two-layer models seem to perform best, we use a three-layer model within Section 2.4.1 to enable more flexibility in examining the partitioning strategy of GIST.

A.1.4 Large-Scale Experiments

Reddit Dataset. For experiments on Reddit, we train 256-dimensional GraphSAGE and GAT models using both GIST and standard, single-GPU methodology. During training, the graph is partitioned into 15 000 sub-graphs. Training would be impossible without such partitioning because the graph is too large to fit into memory. The setting for the number of sub-graphs is the optimal setting proposed in previous work [43]. Models trained using GIST and standard, single-GPU methodologies are compared in terms of F1 score and training time.

All tests are run for 80 epochs with no weight decay, using the Adam optimizer [140]. We find that $\zeta = 500$ achieves consistently high performance for models trained with GIST on Reddit. We adopt a batch size of 10 sub-graphs throughout the training process, which is the optimal setting proposed in previous work [43].

Amazon2M Dataset. For experiments on Amazon2M, we train two to four layer GraphSAGE models with hidden dimensions of 400 and 4 096 using both GIST and standard, single-GPU methodology. We follow the experimental settings of [43]. The training graph is partitioned into 15 000 sub-graphs and a batch size of 10 sub-graphs is used. We find that using $\zeta = 5\,000$ performs consistently well. Models are trained for 400 total epochs with the Adam optimizer [140] and no weight decay.

A.1.5 Training Ultra-Wide GCNs

All settings for ultra-wide GCN experiments in Section 2.4.3 are adopted from the experimental settings of Section 2.4.2; see Appendix A.1.4 for further details. For $d_i > 4\,096$ evaluation must be performed on graph partitions (not the full graph) to avoid memory overflow. As such, the graph is partitioned into 5 000 sub-graphs during testing and F1 score is measured over each partition and averaged. All experiments are performed using a GraphSAGE model, and the hidden dimension of the underlying model is changed between different experiments.

A.1.6 GIST with Layer Sampling

Experiments in Section 2.4.4 adopt the same experimental settings as Section 2.4.2 for the Reddit dataset; see Appendix A.1.4 for further details. Within these experiments, we combine GIST with LADIES [276], a recent layer sampling approach for efficient GCN training. LADIES is used instead of graph partitioning. Any node sampling approach can be adopted—some sampling approach is just needed to avoid memory overflow.

We train 256-dimensional GCN models with either two or three layers. We utilize a vanilla GCN model within this section (as opposed to GraphSAGE or GAT) to simplify the implementation of GIST with LADIES, which creates a disparity in F1 score between the results in Section 2.4.4 and Section 2.4.2. Experiments in Section 2.4.4 compare the performance of the same models trained either with GIST or using standard, single-GPU methodology. In this case, the single-GPU model is just a GCN trained with LADIES.

A.2 GIST vs. Other Distributed Training Methods

Although GIST has been shown to provide benefits in terms of GCN performance and training efficiency in comparison to standard, single-GPU training, other choices for the distributed training of GCNs exist. Within this section, we compare GIST to other natural choices for distributed training, revealing that GCN models trained with GIST achieve favorable performance in comparison to those trained with other common distributed training techniques.

# Machines	Method	F1 Score	Training Time
2	Local SGD	96.37	137.17s
	GIST	**96.40**	**108.67s**
4	Local SGD	95.00	127.63s
	GIST	**96.16**	**116.56s**
8	Local SGD	93.40	129.58s
	GIST	**95.46**	**123.83s**

Table 18: Performance of GraphSAGE models trained using local SGD and GIST on Reddit. We adopt settings described in Section 2.4.2, but use 100 local iterations for both GIST and local SGD.

A.2.1 Local SGD

A simple version of local SGD [158] can be implemented for distributed training of GCNs by training the full model on each separate worker for a certain number of local iterations and intermittently averaging local updates. In comparison to such a methodology, GIST has better computational and communication efficiency because $i)$ it communicates only a small fraction of model parameters to each machine and $ii)$ locally training narrow sub-GCNs is faster than locally training the full model. We perform a direct comparison between local SGD and GIST on the Reddit dataset using a two-layer, 256-dimensional GraphSAGE model; see Table 18. As can be seen, GCN models trained with GIST have lower wall-clock training time and achieve better performance than those trained with local SGD in all cases.

# Machines	Method	F1 Score	Inference Time
2	Ensemble	96.31	3.59s
	GIST	**96.40**	**1.81s**
4	Ensemble	96.10	6.38s
	GIST	**96.16**	**1.81s**
8	Ensemble	95.28	11.95s
	GIST	**95.46**	**1.81s**

Table 19: Performance of GraphSAGE models trained both with GIST and as ensembles of shallow sub-GCNs on Reddit.

A.2.2 Sub-GCN Ensembles

As previously mentioned, increasing the number of local iterations (i.e., ζ in Algorithm 1) decreases communication requirements given a fixed amount of training. When taken to the extreme (i.e., $\zeta \rightarrow \infty$), one could minimize communication requirements by never aggregating sub-GCN parameters, thus forming an ensemble of independently-trained sub-GCNs. We compare GIST to such a methodology[39] in Table 19 using a two-layer, 256-dimensional GraphSAGE model on the Reddit dataset. Though training ensembles

[39]For each sub-GCN, we measure validation accuracy throughout training and add the highest-performing model into the ensemble.

of sub-GCNs minimizes communication, Table 19 reveals that $i)$ models trained with GIST achieve better performance and $ii)$ inference time for sub-GCN ensembles becomes burdensome as the number of sub-GCNs is increased.

A.3 Supplementary Information

All code for this project is publicly-available via github at the following link: `https://github.com/wolfecameron/GIST`

B How much pre-training is enough to discover a good subnetwork?

B.1 Proofs

B.1.1 Convergence Analysis

We consider the following update rule for greedy forward selection, as described in Algorithm 2.

$$\text{(Select new neuron):} \quad \mathbf{q}_k = \underset{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_n)}{\arg\min} \ \ell\left(\tfrac{1}{k} \cdot (\mathbf{z}_{k-1} + \mathbf{q})\right) \tag{23}$$

$$\text{(Add neuron to subnetwork):} \quad \mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{q}_k \tag{24}$$

$$\text{(Uniform average of neuron outputs):} \quad \mathbf{u}_k = \frac{1}{k} \cdot \mathbf{z}_k. \tag{25}$$

Prior to presenting the proofs of the main theoretical results, we introduce several relevant technical lemmas.

Lemma 5. *Consider $\mathbf{u}_k, \mathbf{u}_{k-1} \in \mathcal{M}_N$, representing adjacent iterates of* (23)-(25) *at step k. Additionally, consider an arbitrary update $\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N)$, such that $\mathbf{z}_k = \mathbf{z}_{k-1} + \mathbf{q}$ and $\mathbf{u}_k = \frac{1}{k}\mathbf{z}_k$. Then, we can derive the following expression for the difference between adjacent iterates of* (23)-(25)*:*

$$\mathbf{u}_k - \mathbf{u}_{k-1} = \frac{1}{k}\left(\mathbf{q} - \mathbf{u}_{k-1}\right)$$

Lemma 6. *Because the objective $\ell(\cdot)$, defined over the space $\mathcal{M}_N$, is both quadratic and convex, the following expressions hold for any $\mathbf{s} \in \mathcal{M}_N$:*

$$\begin{aligned} \ell(\mathbf{s}) &\geq \ell(\mathbf{u}_{k-1}) + \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle \\ \ell(\mathbf{s}) &\leq \ell(\mathbf{u}_{k-1}) + \langle \nabla \ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1} \rangle + \|\mathbf{s} - \mathbf{u}_{k-1}\|_2^2 \end{aligned}$$

Observation 1. *From Lemma 6, we can derive the following inequality.*

$$\begin{aligned}
\mathcal{L}_N \geq \mathcal{L}_N^\star &= \min_{\mathbf{s}\in\mathcal{M}_N} \ell(\mathbf{s}) \\
&\geq \min_{\mathbf{s}\in\mathcal{M}_N} \left\{\ell(\mathbf{u}_{k-1}) + \langle \nabla\ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1}\rangle\right\} \\
&= \ell(\mathbf{u}_{k-1}) + \langle \nabla\ell(\mathbf{u}_{k-1}), \mathbf{s}_k - \mathbf{u}_{k-1}\rangle .
\end{aligned}$$

Lemma 7. *Assume there exists a sequence of values $\{z_k\}_{k\geq 0}$ such that $z_0 = 0$ and*

$$|z_{k+1}|^2 \leq |z_k|^2 - 2\beta|z_k| + C, \quad \forall k \geq 0$$

where C and β are positive constants. Then, it must be the case that $|z_k| \leq \max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta}\right)$ for $k > 0$.

Proof. We use an inductive argument to prove the above claim. For the base case of $z_0 = 0$, the claim is trivially true because $|z_0| = 0 \leq \max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta}\right)$. For the inductive case, we define $f(x) = x^2 - 2\beta x + C$. It should be noted that $f(\cdot)$ is a 1-dimensional convex function. Therefore, given some closed interval $[a, b]$ within the domain of f, the maximum value of f over this interval must be achieved on one of the end points. This fact simplifies to the following expression, where a and b are two values within the domain of f such that $a \leq b$.

$$\max_{x\in[a,b]} f(x) = \max\{f(a), f(b)\}$$

From here, we begin the inductive step, for which we consider two cases.

Case 1: Assume that $|z_k| \leq \frac{C}{2\beta}$. Then, the following expression for $|z_{k+1}|$ can be derived.

$$\begin{aligned}
|z_{k+1}|^2 \leq f(|z_k|) \leq \max_z \left\{f(z) : z \in \left[0, \tfrac{C}{2\beta}\right]\right\} &= \max\left\{f(0), f\left(\tfrac{C}{2\beta}\right)\right\} \\
&= \max\left\{C, \frac{C^2}{4\beta^2}\right\} \leq \left[\max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta}\right)\right]^2
\end{aligned}$$

Case 2: Assume that $|z_k| \geq \frac{C}{2\beta}$. Then, the following expression can be derived.

$$|z_{k+1}|^2 \leq |z_k|^2 - 2\beta|z_k| + C \overset{i)}{\leq} |z_k|^2 \leq \left[\max\left(\sqrt{C}, \frac{C}{2}, \frac{C}{2\beta}\right)\right]^2$$

where $i)$ holds because $2\beta|z_k| \geq C$. In both cases, it is shown that $|z_{k+1}| \leq \max\left(\sqrt{C}, \frac{C}{2}, \frac{2}{2\beta}\right)$. Therefore, the lemma is shown to be true by induction. □

Observation 2. *We commonly refer to the value of $\mathcal{L}_N$, representing the quadratic loss of the two-layer* *network over the full dataset. The value of $\mathcal{L}_N$ can be expressed as follows.*

$$\mathcal{L}_N = \ell\left(\frac{1}{N}\sum_{\mathbf{v}\in\texttt{Vert}(\mathcal{M}_N)}\mathbf{v}\right) = \frac{1}{2m}\|f(\mathbf{X},\mathbf{\Theta}) - \mathbf{Y}\|_2^2$$

The expression above is derived by simply applying the definitions associated with $\ell(\cdot)$ that are provided in Section 3.2.

B.1.2 Proof of Lemma 1

We now present the proof of Lemma 1.

Proof. We define $\mathcal{L}_N^\star = \min_{\mathbf{s}\in\mathcal{M}_N} \ell(\mathbf{s})$. Additionally, we define $\mathbf{s}_k$ as follows.

$$\mathbf{s}_k = \arg\min_{\mathbf{s}\in\mathcal{M}_N}\{\langle\nabla\ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1}\rangle\} = \arg\min_{\mathbf{s}\in\texttt{Vert}(\mathcal{M}_N)}\{\langle\nabla\ell(\mathbf{u}_{k-1}), \mathbf{s} - \mathbf{u}_{k-1}\rangle\}.$$

The second equality holds because a linear objective is being optimized on a convex polytope $\mathcal{M}_N$. Thus, the solution to this optimization is known to be achieved on some vertex $\mathbf{v} \in \texttt{Vert}(\mathcal{M}_N)$. Recall, as stated in Section 3.2, that $\mathcal{D}_{\mathcal{M}_N}$ denotes the diameter of the marginal polytope $\mathcal{M}_N$.

We assume the existence of some global two-layer neural network with N hidden neurons from which the pruned network is derived. It should be noted that the N neurons of this global network are used to define the vertices $\mathbf{v} \in \texttt{Vert}(\mathcal{M}_N)$ as described in Section 3.2. As a result, the loss of this global network, which we denote as $\mathcal{L}_N$, is the loss achieved by a uniform average over the N vertices of $\mathcal{M}_N$ (i.e., see (3)). In other words, the loss of the global network at the time of pruning is given by the expression below; see Observation 2.

$$\mathcal{L}_N = \ell\left(\frac{1}{N}\sum_{\mathbf{v}\in\texttt{Vert}(\mathcal{M}_N)}\mathbf{v}\right)$$

It is trivially known that $\mathcal{L}_N \geq \mathcal{L}_N^\star$. Intuitively, the value of $\mathcal{L}_N$ has an implicit dependence on the amount of training underwent by the global network. However, we make no assumptions regarding the global network's training (i.e., $\mathcal{L}_N$ can be arbitrarily large for the purposes of this analysis). Using Observation 1, as well as Lemmas 5 and 6, we derive the following expression for the loss of iterates obtained with (25).

$$\begin{aligned}
\ell(\mathbf{u}_k) &\overset{i)}{=} \min_{\mathbf{q}\in\texttt{Vert}(\mathcal{M}_N)} \ell\left(\frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{q})\right)\\
&\overset{ii)}{\leq} \ell\left(\frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{s}_k)\right)\\
&\overset{iii)}{\leq} \ell(\mathbf{u}_{k-1}) + \left\langle \nabla\ell(\mathbf{u}_{k-1}), \frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{s}_k) - \mathbf{u}_{k-1}\right\rangle + \left\|\frac{1}{k}(\mathbf{z}_{k-1}+\mathbf{s}_k) - \mathbf{u}_{k-1}\right\|_2^2\\
&\overset{iv)}{=} \ell(\mathbf{u}_{k-1}) + \left\langle \nabla\ell(\mathbf{u}_{k-1}), \frac{1}{k}(\mathbf{s}_k - \mathbf{u}_{k-1})\right\rangle + \left\|\frac{1}{k}(\mathbf{s}_k - \mathbf{u}_{k-1})\right\|_2^2\\
&\overset{v)}{\leq} \ell(\mathbf{u}_{k-1}) + \left\langle \nabla\ell(\mathbf{u}_{k-1}), \frac{1}{k}(\mathbf{s}_k - \mathbf{u}_{k-1})\right\rangle + \frac{1}{k^2}\cdot\mathcal{D}^2_{\mathcal{M}_N}\\
&= \ell(\mathbf{u}_{k-1}) + \frac{1}{k}\left\langle \nabla\ell(\mathbf{u}_{k-1}), \mathbf{s}_k - \mathbf{u}_{k-1}\right\rangle + \frac{1}{k^2}\cdot\mathcal{D}^2_{\mathcal{M}_N}\\
&\overset{vi)}{\leq} \left(1-\frac{1}{k}\right)\ell(\mathbf{u}_{k-1}) + \frac{1}{k}\cdot\mathcal{L}_N^\star + \frac{1}{k^2}\cdot\mathcal{D}^2_{\mathcal{M}_N}\\
&\overset{vii)}{\leq} \left(1-\frac{1}{k}\right)\ell(\mathbf{u}_{k-1}) + \frac{1}{k}\cdot\mathcal{L}_N + \frac{1}{k^2}\cdot\mathcal{D}^2_{\mathcal{M}_N}
\end{aligned}$$

where $i)$ is due to (23)-(25), $ii)$ is because $\mathbf{s}_k \in \texttt{Vert}(\mathcal{M}_N)$, $iii)$ is from Lemma 6, $iv)$ is from Lemma 5 since it holds $\frac{1}{k}z_{k-1} - u_{k-1} = -\frac{1}{k}u_{k-1} \Rightarrow u_{k-1} = \frac{1}{k-1}z_{k-1}$, $v)$ is from the definition of $\mathcal{D}_{\mathcal{M}_N}$, and $vi - vii)$ are due to Observation 1. We can then rearrange terms to yield the following recursive expression.

$$\begin{aligned}
\ell(\mathbf{u}_k) \leq \left(1-\frac{1}{k}\right)\ell(\mathbf{u}_{k-1}) + \frac{1}{k}\mathcal{L}_N + \frac{1}{k^2}\mathcal{D}^2_{\mathcal{M}_N} \Rightarrow\\
\ell(\mathbf{u}_k) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}^2_{\mathcal{M}_N} \leq \left(1-\frac{1}{k}\right)\cdot\left(\ell(\mathbf{u}_{k-1}) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}^2_{\mathcal{M}_N}\right)
\end{aligned}$$

By unrolling the recursion in this expression over k iterations, we get the following:

$$\begin{aligned}
\ell(\mathbf{u}_k) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}^2_{\mathcal{M}_N} \leq \prod_{i=2}^{k}\left(1-\frac{1}{i}\right)\cdot\left(\ell(\mathbf{u}_1) - \mathcal{L}_N - \frac{1}{2}\cdot\mathcal{D}^2_{\mathcal{M}_N}\right) \Rightarrow\\
\ell(\mathbf{u}_k) - \mathcal{L}_N - \frac{1}{k}\cdot\mathcal{D}^2_{\mathcal{M}_N} \leq \frac{1}{k}\cdot\left(\ell(\mathbf{u}_1) - \mathcal{L}_N - \frac{1}{2}\cdot\mathcal{D}^2_{\mathcal{M}_N}\right)
\end{aligned}$$

By rearranging terms, we arrive at the following expression

$$\ell(\mathbf{u}_k) \leq \frac{1}{k}\cdot\left(\ell(\mathbf{u}_1) - \mathcal{L}_N + \frac{1}{2}\mathcal{D}^2_{\mathcal{M}_N}\right) + \mathcal{L}_N \leq \mathcal{O}\left(\frac{1}{k}\right) + \mathcal{L}_N \qquad (26)$$

From here, we expand this expression as follows, yielding the final expression from Lemma 1.

$$\begin{aligned}\ell(\mathbf{u}_k) &\leq \frac{1}{k}\left(\ell(\mathbf{u}_1) - \mathcal{L}_N + \frac{1}{2}\mathcal{D}^2_{\mathcal{M}_N}\right) + \mathcal{L}_N \\ &= \frac{1}{k}\left(\ell(\mathbf{u}_1) + \frac{1}{2}\mathcal{D}^2_{\mathcal{M}_N}\right) + \frac{k-1}{k}\mathcal{L}_N \\ &= \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{k}\mathcal{L}_N\end{aligned}$$

□

B.1.3 Proof of Lemma 2

Proof. The proof of Lemma 2 will proceed in two parts. First, we will shows that the iterates derived via Algorithm 3 satisfy Lemma 1. Then, we will derive a bound on the total communication cost across iterations of Algorithm 3.

Convergence Guarantee. Each compute node $v_i \in \mathcal{V}$ maintains an active set of neurons derived from the pruning process with Algorithm 3. We will denote this set of active neurons, which is maintained globally, as $\mathcal{S}$. During each iteration k of Algorithm 3, we compute $q^{(i)}_k = \arg\min_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}^{(i)}_N)} \ell(\frac{1}{k}(\mathbf{z}_{k-1} + \mathbf{q}))$. Notably, this selection process considers the global objective function $\ell(\cdot)$, which is identical to objective used for forward selection in Algorithm 2.

We can compute the value of the global objective function $\ell(\cdot)$ locally for each $v_i \in \mathcal{V}$ by only considering the information present on v_i. To see this, we note that each compute node stores $i)$ a global copy of $\mathbf{z}_{k-1}$, $ii)$ an identical copy of the data and $ii)$ the weights associated with neurons $\mathcal{A}^{(i)}$. As such, we can compute any $\mathbf{q} \in \texttt{Vert}(\mathcal{M}^{(i)}_N)$ as $[\sigma(\mathbf{x}^{(1)}, \boldsymbol{\theta}_j) \; \dots \; \sigma(\mathbf{x}^{(m)}, \boldsymbol{\theta}_j)]/\sqrt{m}$ for $j \in \mathcal{A}^{(i)}$. Then, we can sum this result with the local copy of $\mathbf{z}_{k-1}$ to compute the global objective function $\ell(\cdot)$ on compute node v_i. It should be noted that if we have instead of copy of active neuron weights stored on each compute node, then we could easily similarly reconstruct the vector $\mathbf{z}_{k-1}$ from these weights.

Step I in Algorithm 3 computes a local solution to the global objective $\ell(\cdot)$ over the neurons $\mathcal{A}^{(i)}$ available on compute node v_i. This local solution $\mathbf{q}^{(i)}_k$ is then used to compute the local estimate $\mathbf{z}^{(i)}_k$. From here, Step II compares the best objective values obtained locally on each compute node v_i for all $i \in [V]$ and identifies the node that achieves the best local result. Assuming without loss of generality that the best objective value is achieved on node i, the vector $\mathbf{z}^{(i)}_k$ is then broadcast as the next global value of $\mathbf{z}_k$.

Given that Steps I and II are performed with respect to the global objective loss $\ell(\cdot)$ and considering that $\bigcup_{j=1}^{V} \mathcal{A}^{(j)} = [N]$, each iteration of Algorithm 3 greedily selects the globally best neuron to be included in the active set. As such, the iterates of Algorithm 3 satisfy Lemma 1, yielding an identical convergence rate—in terms of the number of total iterations—between Algorithms 2 and 3.

Communication Guarantee. Currently, Algorithm 3 assumes that we store a global copy of $\mathbf{z}_k$ on each compute node v_i for $i \in [V]$. However, the vector $\mathbf{z}_k \in \mathbb{R}^m$ is quite large in the big data regime, which can lead to increased communication and memory overhead. Alternatively, we could store the weights of neurons in the active set $\{\theta_j : j \in \mathcal{S}\}$. From these weights we could easily perform the following reconstruction.

$$\mathbf{z}_k = \frac{1}{\sqrt{m}} \sum_{j \in \mathcal{S}} [\sigma(\mathbf{x}^{(1)}, \boldsymbol{\theta}_j),\ \sigma(\mathbf{x}^{(2)}, \boldsymbol{\theta}_j),\ \dots\ , \sigma(\mathbf{x}^{(m)},\ \boldsymbol{\theta}_j)]$$

Thus, Algorithm 3 could be modified by broadcasting the neuron weights associated with the greedily selected neuron at each iteration, while still satisfying Lemma 1.

With this in mind, iteration $k \geq 0$ of Algorithm 3 requires that each node $v_j \in \mathcal{V}$ broadcast a constant number of real values, which yields a communication complexity of $\mathcal{O}\left(VB\right)$ per iteration. Additionally, node v_{i_k} must broadcast the selected neuron's weights of size $d+1$, which yields a communication complexity of $\mathcal{O}(Bd)$. As mentioned above, Algorithm 3 will terminate after $k = \mathcal{O}(\frac{1}{\epsilon})$ iterations according to Lemma 1, which yields a total communication complexity of $\mathcal{O}\left((Bd + VB)/\epsilon\right)$. □

B.1.4 Proof of a Faster Rate

Similar to [251], we can obtain a faster $\mathcal{O}(\frac{1}{k^2})$ rate under assumption 1 and the assumption that $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$.

Lemma 8. *Assume that Assumption 1 holds and $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$. Then, the following bound is achieved for two-layer neural networks of width N after k iterations of greedy forward selection:*

$$\ell(\mathbf{u}_k) = \mathcal{O}\left(\frac{1}{k^2 \min(1, \gamma)}\right), \quad \textit{where } \gamma \textit{ is a positive constant.}$$

Proof. We define $\mathbf{w}_k = k(\mathbf{y} - \mathbf{u}_k)$. Also recall that $\ell(\mathbf{z}) = \frac{1}{2}\|\mathbf{z} - \mathbf{y}\|_2^2$. Furthermore, we define $\mathbf{s}_{k+1}$ as follows.

$$\begin{aligned}
\mathbf{s}_{k+1} &= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \nabla \ell(\mathbf{u}_k)^\top (\mathbf{s} - \mathbf{u}_k) \\
&= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \langle \nabla \ell(\mathbf{u}_k), \mathbf{s} - \mathbf{u}_k \rangle \\
&\overset{i)}{=} \arg\min_{\mathbf{s} \in \mathcal{M}_N} \langle \mathbf{u}_k - \mathbf{y}, \mathbf{s} - \mathbf{u_k} \rangle \\
&= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \langle -\mathbf{w}_k, \mathbf{s} - \mathbf{u}_k \rangle \\
&= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \langle \mathbf{w}_k, \mathbf{u}_k - \mathbf{s} \rangle \\
&= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \left\{ \langle \mathbf{w}_k, \mathbf{u}_k \rangle + \langle \mathbf{w_k}, -\mathbf{s} \rangle \right\} \\
&= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \langle \mathbf{w_k}, -\mathbf{s} \rangle \\
&= \arg\min_{\mathbf{s} \in \mathcal{M}_N} \langle \mathbf{w}_k, \mathbf{y} - \mathbf{s} \rangle
\end{aligned}$$

where $i)$ follows from (7). Notice that $\mathbf{s}_k$ minimizes a linear objective (i.e., the dot product with $\mathbf{w}_k$) over the domain of the marginal polytope $\mathcal{M}_N$. As a result, the optimum is achieved on a vertex of the marginal polytope, implying that $\mathbf{s}_k \in \texttt{Vert}(\mathcal{M}_N)$ for all $k > 0$. We assume that $\mathcal{B}(\mathbf{y}, \gamma) \in \mathcal{M}_N$. Under this assumption, it is known that $\mathbf{s}^\star = \mathbf{y} + \gamma \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|_2} \in \mathcal{M}_N$, which allows the following to be derived.

$$\begin{aligned}\langle \mathbf{w}_k, \mathbf{y} - \mathbf{s}_{k+1} \rangle &= \min_{\mathbf{s} \in \mathcal{M}_N} \langle \mathbf{w}_k, \mathbf{y} - \mathbf{s} \rangle \\ &\leq \langle \mathbf{w}_k, \mathbf{y} - \mathbf{s}^\star \rangle \\ &= -\gamma \|\mathbf{w}_k\|_2 \end{aligned} \tag{27}$$

From (25), the following expressions for $\mathbf{u}_k$ and $\mathbf{q}_k$ can be derived.

$$\mathbf{u}_k = \frac{1}{k} \cdot \mathbf{z}_k = \frac{1}{k} \cdot [(k-1)\mathbf{u}_{k-1} + \mathbf{q}_k] \tag{28}$$

$$\begin{aligned}\mathbf{q}_k &= \operatorname*{arg\,min}_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N)} \ell\left(\frac{1}{k}[\mathbf{z}_{k-1} + \mathbf{q}]\right) \\ &= \operatorname*{arg\,min}_{\mathbf{q} \in \texttt{Vert}(\mathcal{M}_N)} \left\| \frac{1}{k}[(k-1)\mathbf{u}_{k-1} + \mathbf{q}] - \mathbf{y} \right\|_2^2 \end{aligned} \tag{29}$$

Combining all of this together, the following expression can be derived for $\|\mathbf{w}_k\|_2^2$, where $\mathcal{D}_{\mathcal{M}_N}$ is the diameter of $\mathcal{M}_N$:

$$\begin{aligned}\|\mathbf{w}_k\|_2^2 &= \|k(\mathbf{y} - \mathbf{u}_k)\|_2^2 = \|k(\mathbf{u}_k - \mathbf{y})\|_2^2 \\ &\overset{i)}{=} \min_{\mathbf{q} \in \mathcal{M}_N} \|k(\tfrac{1}{k}[(k-1)\mathbf{u}_{k-1} + \mathbf{q}] - \mathbf{y})\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \|(k-1)\mathbf{u}_{k-1} + \mathbf{q} - k\mathbf{y}\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \| - (k-1)\mathbf{y} + (k-1)\mathbf{u}_{k-1} + \mathbf{q} - \mathbf{y}\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \| - \mathbf{w}_{k-1} - \mathbf{y} + \mathbf{q}\|_2^2 \\ &= \min_{\mathbf{q} \in \mathcal{M}_N} \|\mathbf{w}_{k-1} + \mathbf{y} - \mathbf{q}\|_2^2 \\ &\overset{ii)}{\leq} \|\mathbf{w}_{k-1} + \mathbf{y} - \mathbf{s}_k\|_2^2 \\ &= \|\mathbf{w}_{k-1}\|_2^2 + 2\langle \mathbf{w}_{k-1}, \mathbf{y} - \mathbf{s}_k \rangle + \|\mathbf{y} - \mathbf{s}_k\|_2^2 \\ &\leq \|\mathbf{w}_{k-1}\|_2^2 + 2\langle \mathbf{w}_{k-1}, \mathbf{y} - \mathbf{s}_k \rangle + \mathcal{D}_{\mathcal{M}_N}^2 \\ &\overset{iii)}{\leq} \|\mathbf{w}_{k-1}\|_2^2 - 2\gamma\|\mathbf{w}_{k-1}\|_2 + \mathcal{D}_{\mathcal{M}_N}^2 \end{aligned}$$

where $i)$ follows from (28) and (29), $ii)$ follows from the fact that $\mathbf{s}_k \in \mathcal{M}_N$, and $iii)$ follows from (27). Therefore, from this analysis, the following recursive expression for the value of $\|\mathbf{w}_k\|^2$ is derived:

$$\|\mathbf{w}_k\|_2^2 \leq \|\mathbf{w}_{k-1}\|_2^2 - 2\gamma\|\mathbf{w}_{k-1}\|_2 + \mathcal{D}_{\mathcal{M}_N}^2 \tag{30}$$

Then, by invoking Lemma 7, we derive the following inequality.

$$\begin{aligned}\|\mathbf{w}_k\|_2^2 &\leq \max\left\{\mathcal{D}_{\mathcal{M}_n}, \frac{\mathcal{D}_{\mathcal{M}_N}^2}{2}, \frac{\mathcal{D}_{\mathcal{M}_N}^2}{2\gamma}\right\} \\ &= \mathcal{O}\left(\frac{1}{\min(1,\gamma)}\right)\end{aligned}$$

With this in mind, the following expression can then be derived for the loss $\ell(\mathbf{u}_k)$ achieved by (23)-(25) after k iterations.

$$\ell(\mathbf{u}_k) = \frac{1}{2}\|\mathbf{u}_k - \mathbf{y}\|_2^2 = \frac{\|\mathbf{w}_k\|_2^2}{2k^2} = \mathcal{O}\left(\frac{1}{k^2\min(1,\gamma)^2}\right)$$

This yields the desired expression, thus completing the proof. □

B.1.5 Training Analysis

Prior to analyzing the amount of training needed for a good pruning loss, several supplemental theorems and lemmas exist that must be introduced. From [191], we utilize theorems regarding the convergence rates of two-layer neural networks trained with GD and SGD. We begin with the theorem for the convergence of GD in Theorem 7, then provide the associated convergence rate for SGD within Theorem 8. Both Theorems 7 and 8 are simply restated from [191] for convenience purposes.

Theorem 7. *[191]. Assume there exists a two-layer neural network and associated dataset as described in Section 3.2. Denote N as the number of hidden neurons in the two-layer neural network, m as the number of unique examples in the dataset, and d as the input dimension of examples in the dataset. Assume without loss of generality that the input data within the dataset is normalized so that $\|\mathbf{x}^{(i)}\|_2 = 1$ for all $i \in [m]$. A moderate amount of overparameterization within the two-layer network is assumed, given by $Nd > m^2$. Furthermore, it is assumed that $m > d$ and that the first and second derivatives of the network's activation function are bounded (i.e., $|\sigma'_+(\cdot)| \leq \delta$ and $|\sigma''_+(\cdot)| \leq \delta$ for some $\delta \in \mathbb{R}$). Given these assumptions, the following bound is achieved with a high probability by training the neural network with gradient descent.*

$$\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2 \leq \left(1 - c\frac{d}{m}\right)^t \cdot \|f(\mathbf{X}, \mathbf{\Theta}_0) - \mathbf{Y}\|_2 \tag{31}$$

In (31), *$c \in \mathbb{R}$, $\mathbf{\Theta}_t = \{\boldsymbol{\theta}_{1,t}, \dots, \boldsymbol{\theta}_{N,t}\}$ represents the network weights at iteration t of gradient descent, $f(\cdot,\cdot) \in \mathbb{R}^m$ represents the network output over the full dataset $\mathbf{X} \in \mathbb{R}^{m \times d}$, and $Y = [y^{(1)}, \dots, y^{(m)}]^T$ represents a vector of all dataset labels. $\mathbf{\Theta}_0$ is assumed to be randomly sampled from a normal distribution (i.e., $\mathbf{\Theta}_0 \sim \mathcal{N}(0,1)$).*

Theorem 8. *[191]. Here, all assumptions of Theorem 7 are adopted, but we assume the two-layer neural network is trained with SGD instead of GD. For SGD, parameter updates are performed over a sequence of randomly-sampled examples within the training dataset (i.e., the true gradient is not computed for each update). Given the assumptions, there exists some event E with probability $P[E] \geq \max\left(\kappa,\ 1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)$,* *where $c_1, c_2, \kappa \in \mathbb{R}$, $0 \leq c_1 \leq \frac{4}{3}$, and $0 < \kappa \leq 1$. Given the event E, with high probability the following bound is achieved for training a two-layer neural network with SGD.*

$$\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right] \leq \left(1 - c\frac{d}{m^2}\right)^t \cdot \|f(\mathbf{X}, \mathbf{\Theta}_0) - \mathbf{Y}\|_2^2 \tag{32}$$

In (32), *$c \in \mathbb{R}$, $\mathbf{\Theta}_t$ represent the network weights at iteration t of SGD, $\mathbb{1}_E$ is the indicator function for event E, $f(\cdot, \cdot)$ represents the output of the two layer neural network over the entire dataset $\mathbf{X} \in \mathbb{R}^{m \times d}$, and $\mathbf{Y} \in \mathbb{R}^m$ represents a vector of all labels in the dataset.*

It should be noted that the overparameterization assumptions within Theorems 7 and 8 are very mild, which leads us to adopt this analysis within our work. Namely, we only require that the number of examples in the dataset exceeds the input dimension and the number of parameters within the first neural network layer exceeds the squared size of the dataset. In comparison, previous work lower bounds the number of hidden neurons in the two-layer neural network (i.e., more restrictive than the number of parameters in the first layer) with higher-order polynomials of m to achieve similar convergence guarantees [8, 60, 151].

In comparing the convergence rates of Theorems 7 and 8, one can notice that these linear convergence rates are very similar. The extra factor of m within the denominator of Theorem 8 is intuitively due to the fact that m updates are performed in a single pass through the dataset for SGD, while GD uses the full dataset at every parameter update. Such alignment between the convergence guarantees for GD and SGD allows our analysis to be similar for both algorithms.

B.1.6 Proof of Theorem 1

We now provide the proof for Theorem 1.

Proof. From Theorem 8, we have a bound for $\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right]$, where $\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2$ represents the loss over the entire dataset after t iterations of SGD (i.e., without the factor of $\frac{1}{2}$). Two sources of stochasticity exist within the expectation expression $\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right]$: $i)$ randomness over the event E and $ii)$ randomness over the t-th iteration of SGD given the first $t-1$ iterations. The probability of event E is independent of the randomness over SGD iterations, which allows the following expression to be derived.

$$\begin{aligned}\mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \mathbb{1}_E\right] &\overset{i)}{=} \mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right] \cdot \mathbb{E}\left[\mathbb{1}_E\right] \\ &\overset{ii)}{\geq} \max\left(\kappa,\ 1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right) \mathbb{E}\left[\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2\right]\end{aligned}$$

where $i)$ holds from the independence of expectations and $ii)$ is derived from the probability expression for event E in Theorem 8. Notably, the expectation within the above expression now has only a single source of stochasticity—the randomness over SGD iterations. Combining the above expression with (32) from Theorem 8 yields the following, where two possible cases exist.

Case 1: $\max\left(\kappa,\ 1 - c_1(c_2\sqrt{\frac{m}{d}})\right) = 1 - c_1(c_2\sqrt{\frac{m}{d}})$

$$\left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)\mathbb{E}\left[\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\boldsymbol{\Theta}_0) - \mathbf{Y}\|_2^2 \Rightarrow$$

$$\mathbb{E}\left[\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)^{-1}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\boldsymbol{\Theta}_0) - \mathbf{Y}\|_2^2 \tag{33}$$

Case 2: $\max\left(\kappa,\ 1 - c_1(c_2\sqrt{\frac{m}{d}})\right) = \kappa$

$$\kappa \cdot \mathbb{E}\left[\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\boldsymbol{\Theta}_0) - \mathbf{Y}\|_2^2 \Rightarrow$$

$$\mathbb{E}\left[\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right] \leq \kappa^{-1}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\boldsymbol{\Theta}_0) - \mathbf{Y}\|_2^2 \tag{34}$$

From here, we use Observation 2 to derive the following, where the expectation is with respect to randomness over SGD iterations (i.e., we assume the global two-layer network of width N is trained with SGD).

$$\begin{aligned}\mathbb{E}[\mathcal{L}_N] &= \mathbb{E}\left[\frac{1}{2m}\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right] \\ &\overset{i)}{=} \frac{1}{2m}\mathbb{E}\left[\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right]\end{aligned}$$

Here, the equality in $i)$ holds true because $\frac{1}{2m}$ is a constant value given a fixed dataset. Now, notice that this expectation expression $\mathbb{E}\left[\|f(\mathbf{X},\boldsymbol{\Theta}_t) - \mathbf{Y}\|_2^2\right]$ is identical to the expectation within (33)-(34) (i.e., both expectations are with respect to randomness over SGD iterations). Thus, the above expression can be combined with (33) and (34) to yield the following.

$$\mathbb{E}[\mathcal{L}_N] \leq \frac{1}{2m}\left[\max\left(\kappa, \left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)\right)\right]^{-1}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\boldsymbol{\Theta}_0) - \mathbf{Y}\|_2^2 \tag{35}$$

Then, we can substitute (35) into Lemma 1 to derive the final result, where expectations are with respect to randomness over SGD iterations. We also define $\mathcal{L}_0 = \|f(\mathbf{X},\Theta_0) - \mathbf{Y}\|_2^2$ and $\zeta = \left[\max\left(\kappa, \left(1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}\right)\right)\right]^{-1} > 0$.

$$\begin{aligned}\mathbb{E}[\ell(\mathbf{u}_k)] &\leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{k-1}{k}\mathbb{E}[\mathcal{L}_N] \\ &\leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \|f(\mathbf{X},\boldsymbol{\Theta}_0) - \mathbf{Y}\|_2^2 \\ &= \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \mathcal{L}_0\end{aligned} \tag{36}$$

□

B.1.7 Proof of Theorem 2

We now provide the proof for Theorem 2.

Proof. We begin with (36) from the proof of Theorem 1.

$$\mathbb{E}[\ell(\mathbf{u}_k)] \leq \frac{1}{k}\mathbb{E}[\ell(\mathbf{u}_1)] + \frac{1}{2k}\mathbb{E}[\mathcal{D}^2_{\mathcal{M}_N}] + \frac{(k-1)\zeta}{2mk}\left(1 - c\frac{d}{m^2}\right)^t \mathcal{L}_0$$

It can be seen that all terms on the right-hand-side of the equation above will decay to zero as k increases aside from the rightmost term. The rightmost term of (36) will remain fixed as k increases due to its factor of $k-1$ in the numerator.

Within (36), there are two parameters that can be modified by the practitioner: N (i.e., ζ depends on N) and t. All other factors within the expressions are constants based on the dataset that cannot be modified. N only appears in the $1 - c_1\left(c_2\sqrt{\frac{m}{d}}\right)^{\frac{1}{Nd}}$ factor of ζ, thus revealing that the value of N cannot be naively increased within (36) to remove the factor of $k-1$.

To determine how t can be modified to achieve a better pruning rate, we notice that a setting of $\left(1 - c\frac{d}{m^2}\right)^t = \mathcal{O}\left(\frac{1}{k}\right)$ would cancel the factor of $k-1$ in (36). With this in mind, we observe the following.

$$\begin{aligned}
\left(1 - c\frac{d}{m^2}\right)^t &= O\left(\frac{1}{k}\right) \Rightarrow \\
t \cdot \log\left(1 - c\frac{d}{m^2}\right) &= O(-\log(k)) \Rightarrow \\
t &= \frac{O(-\log(k))}{\log(1 - c\frac{d}{m^2})} \Rightarrow \\
t &\approx O\left(\frac{-\log(k)}{\log(1 - c\frac{d}{m^2})}\right)
\end{aligned} \tag{37}$$

If the amount of training in (37) is satisfied, the factor of $k-1$ in the rightmost term in Eq. (36) will be canceled, causing the expected pruning loss to decay to zero as k increases. Based on Theorem 8, it must be the case that $(1 - c\frac{d}{m^2}) \in [0, 1]$ in order for SGD to converge. As a result, $\lim_{t\to\infty}(1 - c\frac{d}{m^2})^t = 0$, causing the rightmost term in (36) to approach zero as $t \to \infty$. In other words, the value of t can increase beyond the bound in (37) without damaging the loss of the pruned network, as the rightmost term in (36) will simply become zero. This observation that t can increase beyond the value in (37) yields the final expression for the number of SGD iterations required to achieve the desired $\mathcal{O}(\frac{1}{k})$ pruning rate.

$$t \gtrapprox O\left(\frac{-\log(k)}{\log(1 - c\frac{d}{m^2})}\right)$$

$\square$

B.1.8 Supplemental Result and Proof with GD

In addition to the proof of Theorem 2, we provide a similar result for neural networks trained with GD to further support our analysis of the amount of training required to achieve good loss via pruning.

Theorem 9. *Assume a two-layer neural network of width N was trained for t iterations with gradient descent over a dataset of size m. Furthermore, assume that $Nd > m^2$ and $m > d$, where d represents the input dimension of data in D. When the network is pruned via* (23)-(25)*, it will achieve a loss $\mathcal{L}' \propto O(\frac{1}{k})$ if the number of gradient descent iterations t satisfies the following condition.*

$$t \gtrapprox O\left(\frac{-\log(k)}{\log\left(1 - c\frac{d}{m}\right)}\right) \tag{38}$$

Otherwise, the loss of the pruned network will not improve during successive iterations of (23)-(25).

Proof. Beginning with Lemma 1, we can use Theorem 7 to arrive at the following expression. Here, we define $\mathcal{L}_0 = \|f(\mathbf{X}, \mathbf{\Theta}_0) - \mathbf{Y}\|_2^2$ and define $\mathcal{L}_N$ as described in Observation 2.

$$\begin{aligned} l(\mathbf{u}_k) &\leq \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{k}\mathcal{L}_N \\ &\overset{i)}{=} \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{2mk}\|f(\mathbf{X}, \mathbf{\Theta}_t) - \mathbf{Y}\|_2^2 \\ &\overset{ii)}{\leq} \frac{1}{k}\ell(\mathbf{u}_1) + \frac{1}{2k}\mathcal{D}^2_{\mathcal{M}_N} + \frac{k-1}{2k}(1 - c\frac{d}{m})^{2t} \cdot \mathcal{L}_0 \end{aligned} \tag{39}$$

where $i)$ is due to Observation 2 and $ii)$ is from Theorem 7. From this expression, one can realize that an $O(\frac{1}{k})$ rate is only achieved if the factor of $k-1$ within the rightmost term of (39) is removed. This factor, if not counteracted, will cause the upper bound for $\ell(\mathbf{u^k})$ to remain constant, as the right hand expression of (39) would be dominated by the value of $\mathcal{L}_0$. However, this poor upper bound can be avoided by manipulating the $(1 - c\frac{d}{m})^{2t}$ term to eliminate the factor of $k-1$. For this to happen, it must be true that $(1 - c\frac{d}{m})^{2t} \approx O(\frac{1}{k})$, which allows the following asymptotic expression for t to be derived.

$$\begin{aligned} \left(1 - c\frac{d}{m}\right)^{2t} &= O(\frac{1}{k}) \Rightarrow \\ 2t \cdot \log\left(1 - c\frac{d}{m}\right) &= O(-\log(k)) \Rightarrow \\ t &= \frac{O(-\log(k))}{2 \cdot \log\left(1 - c\frac{d}{m}\right)} \Rightarrow \\ t &\approx O\left(\frac{-\log k}{\log\left(1 - c\frac{d}{m}\right)}\right) \end{aligned} \tag{40}$$

If the amount of training in (40) is satisfied, it allows the factor of $k-1$ in the rightmost term of 39 to be canceled. It is trivially true that $\lim_{t\to\infty}(1 - c\frac{d}{m})^{2t} = 0$ because $(1 - c\frac{d}{m}) \in [0, 1]$ if gradient descent converges; see Theorem 7. As a result, the rightmost term in (39) also approaches zero as t increases, allowing the $O(\frac{1}{k})$ pruning rate to be achieved. This observation that t can be increased beyond the value in (40) without issues

m	MNIST	CIFAR10
1000	1	4
2000	1	7
3000	1	6
4000	2	9
5000	2	10

Table 20: Results of empirically analyzing whether $\mathbf{y} \in \mathcal{M}_N$. For each of the possible datasets and sizes, we report the number of training epochs required before the assumption was satisfied for the best possible learning rate setting.

leads to the final expression from Theorem 9.

$$t \gtrapprox O\left(\frac{-\log(k)}{\log\left(1 - c\frac{d}{m}\right)}\right)$$

□

B.2 Empirical Analysis of $\mathbf{y} \in \mathcal{M}_N$

To achieve the faster rate provided in Lemma 8, we make the assumption that $B(\mathbf{y}, \gamma) \in \mathcal{M}_N$. If it is assumed that $\gamma > 0$, this assumption is slightly stronger than $\mathbf{y} \in \mathcal{M}_N$, as it implies $\mathbf{y}$ cannot lie on the perimeter of $\mathcal{M}_N$. However, this assumption roughly requires that $\mathbf{y} \in \mathcal{M}_N$. Although previous work has attempted to analyze this assumption theoretically [251], it is not immediately clear whether this assumption is reasonable in practice.

We perform experiments using two-layer neural networks trained on different image classification datasets to determine if this assumption is satisfied in practice. Namely, we train a two-layer neural network on uniformly downsampled (i.e., equal number of examples taken from each class) versions of the MNIST and CIFAR10 datasets and test the $\mathbf{y} \in \mathcal{M}_N$ condition following every epoch (i.e., see Section B.2.2 for details). We record the first epoch of training after which $\mathbf{y} \in \mathcal{M}_N$ is true and report the results in Table 20. As can be seen, *the assumption is never satisfied at initialization and tends to require more training for larger datasets*.

B.2.1 Models and Datasets

All tests were performed with a two-layer neural network as defined in Section 3.2. This model contains a single output neuron, N hidden neurons, and uses a smooth activation function (i.e., we use the sigmoid activation). For each test, we ensure the overparameterization requirements presented in Theorem 2 (i.e., $Nd > m^2$ and $m > d$, where N is the number of hidden neurons, d is the input dimension, and m is the size of the dataset) are satisfied. For simplicity, the network is trained without any data augmentation, batch normalization, or dropout.

Experiments are performed using the MNIST and CIFAR10 datasets. These datasets are reduced in size in order to make overparameterization assumptions within the two-layer neural network easier to satisfy. In particular, we perform tests with 1000, 2000, 3000, 4000, and 5000 dataset examples for each of the separate

m	N
1000	1300
2000	6000
3000	12000
4000	21000
5000	32000

Table 21: Displays the hidden dimension of the two-layer neural network used to test the $\mathbf{y} \in \mathcal{M}_N$ assumption for different dataset sizes. Hidden dimensions are the same between tests with MNIST and CIFAR10.

datasets, where each dataset size is constructed by randomly sampling an equal number of data examples form each of the ten possible classes. Additionally, CIFAR10 images are downsampled from a spatial resolution of 32×32 to a spatial resolution of 18×18 using bilinear interpolation to further ease overparameterization requirements. Dataset examples were flattened to produce a single input vector for each image, which can be passed as input to the two-layer neural network.

The number of hidden neurons utilized for tests with different dataset sizes m are shown in Table 21, where the selected value of N is (roughly) the smallest round value to satisfy overparameterization assumptions. The hidden dimension of the two-layer network was kept constant between datasets because the CIFAR10 images were downsampled such that the input dimension is relatively similar to MNIST. To improve training stability, we add a sigmoid activation function to the output neuron of the two-layer network and train the network with binary cross entropy loss. The addition of this output activation function slightly complicates the empirical determination of whether $\mathbf{y} \in \mathcal{M}_N$, which is further described in Section B.2.2. Despite slightly deviating from the setup in Section 3.2, this modification greatly improves training stability and is more realistic (i.e., classification datasets are not regularly trained with ℓ_2 regression loss as described in Section 3.2). The network is trained using stochastic gradient descent.[40] We adopt a batch size of one to match the stochastic gradient descent setting exactly and do not use any weight decay.

B.2.2 Determining Convex Hull Membership

As mentioned in Section 3.2, we define $\mathbf{y} = \left[y^{(1)}, y^{(2)}, \dots, y^{(m)}\right] / \sqrt{m}$ as a vector containing all labels in our dataset (i.e., in this case, a vector of binary labels). Furthermore, we define $\phi_{i,j} = \sigma(\mathbf{x}^{(j)}, \boldsymbol{\theta}_i)$ as the output of neuron i for example j in the dataset. These values can be concatenated as $\mathbf{\Phi}_i = [\phi_{i,1}, \phi_{i,2}, \dots, \phi_{i,m}] / \sqrt{m}$, forming a vector of output activations for each neuron over the entire dataset. Then, we define $\mathcal{M}_N = \texttt{Conv}\left\{\mathbf{\Phi}_i : i \in [N]\right\}$. It should be noted that the activation vectors $\mathbf{\Phi}_i$ are obtained from the N neurons of a two-layer neural network that has been trained for any number of iterations T. For the experiments in this section, the vectors $\mathbf{\Phi}_i$ are computed after each epoch of training to determine whether $\mathbf{y} \in \mathcal{M}_N$, and we report the number for the first epoch in which this membership is satisfied.

The convex hull membership problem can be formulated as a linear program, and we utilize this formulation to empirically determine when $\mathbf{y} \in \mathcal{M}_N$. In particular, we consider the following linear program.

[40]Experiments are repeated with multiple learning rates and the best results are reported.

$$\begin{aligned} &\min \mathbf{c}^\top \boldsymbol{\alpha} \\ \text{s.t. } &A\boldsymbol{\alpha} = \mathbf{y} \\ &Z\boldsymbol{\alpha} = \mathbf{1} \\ &\boldsymbol{\alpha} \geq \mathbf{0} \end{aligned} \tag{41}$$

where $\mathbf{c} \in \mathbb{R}^N$ is an arbitrary cost vector, $A = (\boldsymbol{\Phi}_1 \dots \boldsymbol{\Phi}_N) \in \mathbb{R}^{m \times N}$, and $Z = (1 \dots 1) \in \mathbb{R}^{1 \times N}$, and $\boldsymbol{\alpha}$ is the simplex being optimized within the linear program. If a viable solution to the linear program in (41) can be found, then it is known that $\mathbf{y} \in \mathcal{M}_N$. Therefore, we empirically solve for the vectors $\mathbf{y}$ and $(\boldsymbol{\Phi}_i)_{i=1}^N$ using the two-layer neural network after each epoch and use the `linprog` package in SciPy to determine whether (41) is solvable, thus allowing us to determine when $\mathbf{y} \in \mathcal{M}_N$

In our actual implementation, we slightly modify the linear program in (41) to account for the fact that our two-layer neural network is trained with a sigmoid activation on the output neuron. Namely, one can observe that for targets of one, the neural network output can be considered correct if the output value is greater than zero and vice versa. We formulate the following linear program to better reflect this correct classification behavior:

$$\begin{aligned} &\min \mathbf{c}^\top \boldsymbol{\alpha} \\ \text{s.t. } &A\boldsymbol{\alpha} < \mathbf{0} \\ &B\boldsymbol{\alpha} > \mathbf{0} \\ &Z\boldsymbol{\alpha} = \mathbf{1} \\ &\boldsymbol{\alpha} \geq \mathbf{0} \end{aligned} \tag{42}$$

where $A \in \mathbb{R}^{\frac{m}{2} \times N}$ is a matrix of feature column vectors corresponding to a label of zero, $B \in \mathbb{R}^{\frac{m}{2} \times N}$ is a matrix of feature column vectors corresponding to a label of one, and other variables are defined as in (41). In words, (42) is solvable whenever a simplex can be found to re-weight neuron outputs such that the correct classification is produced for all examples in the dataset. The formulation in (42) better determines whether $\mathbf{y} \in \mathcal{M}_N$ given the modification of our two-layer neural network to solve a binary classification problem.

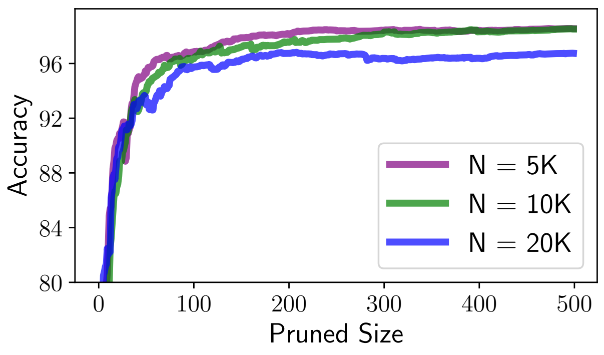

Figure 25: The performance of two-layer networks with different hidden dimensions over the entire, binarized MNIST dataset that have been pruned to various different hidden dimensions.

B.3 Experimental Details

B.3.1 Two-layer Network Experiments

Algorithm 6: Greedy Forward Selection for Two-Layer Network

```
$N$ := hidden size; $P$ := pruned size; $\mathcal{D}$ := training dataset
$\boldsymbol{\Theta}$ := weights; $\boldsymbol{\Theta}^\star = \varnothing$
$i = 0$
while $i < P$ do
    $j = 0$
    best_idx $= -1$
    $\mathcal{L}^\star = \infty$
    $\boldsymbol{X}', \mathbf{y}' \sim \mathcal{D}$
    while $j < N$ do
        $\mathcal{L}_j = \frac{1}{2}\left(f_{(\boldsymbol{\Theta}^\star \cup j)}(\mathbf{X}') - \mathbf{y}'\right)^2$
        if $\mathcal{L}_j < \mathcal{L}^\star$ then
            $\mathcal{L}^\star = \mathcal{L}_j$
            best_idx $= j$
        end
        $j = j + 1$
    end
    $\boldsymbol{\Theta}^\star = \boldsymbol{\Theta}^\star \cup$ best_idx
    $i = i + 1$
end
return $\boldsymbol{\Theta}^\star$
```

We first provide a clear algorithmic description of the pruning algorithm used within all two-layer network experiments. This algorithm closely reflects the update rule provided in (10). However, instead of measuring loss over the full dataset to perform greedy forward selection, we perform selection with respect to the loss over a single mini-batch to improve the efficiency of the pruning algorithm. We use $f(\boldsymbol{\Theta}, \boldsymbol{X}')$ to denote the output of a two-layer network network with parameters $\boldsymbol{\Theta}$ over the mini-batch $\boldsymbol{X}'$ and $f_S(\boldsymbol{\Theta}, \boldsymbol{X}')$ to denote

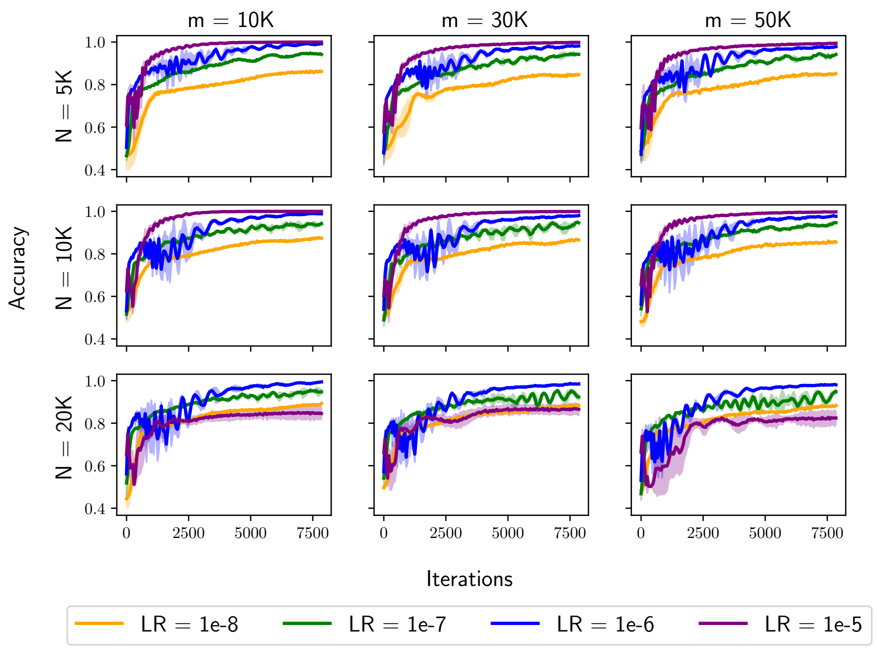

Figure 26: Validation accuracy of two-layer networks on binarized versions of MNIST as described in the Two-layer Network Experiments Section in the main text. Different subplots display results for several different learning rates for models trained with different settings of m and N (i.e., dataset size and number of hidden neurons, respectively). Shaded regions represent standard deviations of results, recorded across three separate trials.

the same output only considering the neuron indices included within the set $\mathcal{S}$. *Notice that a single neuron can be selected more than once during pruning*.

We now present in-depth details regarding the hyperparameters that were selected for the two-layer network experiments in the main text. To tune hyperparameters, we perform a random 80-20 split to generate a validation set. Experiments are repeated three times for each hyperparameter setting, and hyperparameters that yield the best average validation performance are selected.

Models are trained using SGD with momentum of 0.9 and no weight decay. Perturbing weight decay and momentum hyperparameters does not meaningfully impact performance, leading us to maintain this setting in all experiments with two-layer networks. We also use a batch size of 128, which was the largest size that could fit in the memory of our GPU. To determine the optimal learning rate and number of pre-training iterations, we adopt the same setup as described in the Two-layer Network Experiments section in the main text and train two-layer networks of various sizes with numerous different learning rates. These experiments are then replicated across several different sub-dataset sizes. The results of these experiments are shown in Figure 26. From these results, it can be seen that the optimal learning rate does not change with the size of the dataset, but it does depend on the size of the network. Namely, for two-layer networks with 5K or 10K hidden neurons the optimal learning rate is 1e-5, while for two-layer networks with 20K hidden neurons the optimal learning rate is 1e-6. It can also be seen in Figure 26 that models converge in roughly 8000 training iterations for all experimental settings, which leads us to adopt this amount of pre-training in the main experiments.

To determine the optimal pruned model size, we first fully pre-train two-layer networks of different hidden sizes (i.e., $N \in \{5K, 10K, 20K\}$) over the full, binarized MNIST dataset. Then, these models are pruned to different hidden neuron sizes between 1 and 500. At each possible hidden dimension for the pruned model, we measure the performance of the pruned model over the entire training dataset. The results of this experiment are depicted in Figure 25. As can be seen, the accuracy of the pruned models plateaus at a hidden dimension of 200 (roughly). As a result, we adopt a size of 200 neurons as our pruned model size within all two-layer network experiments.

B.3.2 CNN Experiments

Algorithm 7: Greedy Forward Selection for Deep CNN

```
$C$ := hidden sizes; $\epsilon$ := stopping criterion; $\mathcal{D}$ := training dataset
$\mathbf{\Theta}$ := weights; $L$ := # Layers; $\mathbf{\Theta}^\star_\ell = \varnothing \ \forall \ell \in [L]$
$\ell = 0$
while $\ell < L$ do
    while convergence criterion is not met do
        $j = 0$
        best_idx $= -1$
        $\mathcal{L}^\star = \infty$
        $\boldsymbol{X}', \mathbf{y}' \sim \mathcal{D}$
        while $j < C_\ell$ do
            $\mathbf{\Theta}' = \mathbf{\Theta}^\star \cup \mathbf{\Theta}_{\ell:} \cup (\mathbf{\Theta}^\star_\ell \cup j)$
            $\mathcal{L}_{\ell,j} = \frac{1}{2}\left(f_{\mathbf{\Theta}'}(\mathbf{X}') - \mathbf{y}'\right)^2$
            if $\mathcal{L}_{\ell,j} < \mathcal{L}^\star$ then
                $\mathcal{L}^\star = \mathcal{L}_{\ell,j}$
                best_idx $= j$
            end
            $j = j + 1$
        end
        $\mathbf{\Theta}^\star_\ell = \mathbf{\Theta}^\star_\ell \cup$ best_idx
    end
    $\ell = \ell + 1$
end
return $\mathbf{\Theta}^\star$
```

We begin with an in-depth algorithmic description of the greedy forward selection algorithm that was used for structured, channel-based pruning of multi-layer CNN architectures. This algorithm is identical to the greedy forward selection algorithm adopted in [251]. In this algorithm, we denote the weights of the deep network as as $\mathbf{\Theta}$, and reference the weights within layer ℓ of the network as $\mathbf{\Theta}_\ell$. Similarly, we use $\mathbf{\Theta}_{\ell:}$ to denote the weights of all layers following layer ℓ and $\mathbf{\Theta}_{:\ell}$ to denote the weights of all layers up to and including layer ℓ. C denotes a list of hidden sizes within the network, where C_ℓ denotes the number of channels within layer ℓ of the CNN. Again, we use $f(\mathbf{\Theta}, \boldsymbol{X}')$ to denote the output of a two-layer network with parameters $\mathbf{\Theta}$ over

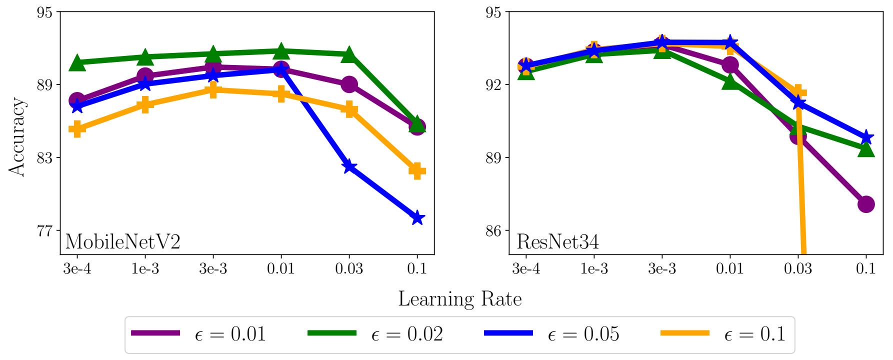

Figure 27: Subnetwork validation accuracy on the CIFAR10 dataset for different settings of ϵ and initial learning rate for fine-tuning. All models are pre-trained identically for 200 epochs. Fine-tuning is performed for 80 epochs, and we report validation accuracy for each subnetwork at the end of fine-tuning.

the mini-batch $\boldsymbol{X}'$ and $f_{\mathcal{S}}(\boldsymbol{\Theta}, \boldsymbol{X}')$ to denote the same output only considering the channel indices included within the set $\mathcal{S}$.

Now, we present more details regarding the hyperparameters that were utilized within large-scale experiments. For ImageNet experiments, we adopt the settings of [251].[41] For CIFAR10, however, we tune both the setting of ϵ and the initial learning rate for fine-tuning using a grid search for both MobileNetV2 and ResNet34 architectures. This grid search is performed using a validation set on CIFAR10, constructed using a random 80-20 split on the training dataset. Optimal hyperparameters are selected based on their performance on the validation set. The results of this grid search are shown in Figure 27. As can be seen, for MobileNetV2, the best results are achieved using a setting of $\epsilon = 0.02$, which results in a subnetwork with 60% of the FLOPS of the dense model. Furthermore, an initial learning rate of 0.01 yields consistent subnetwork performance for MobileNetV2. For ResNet34, a setting of $\epsilon = 0.05$ yields the best results and yields a subnetwork with 60% of the FLOPS of the dense model. Again, an initial learning rate of 0.01 for fine-tuning yields the best results for ResNet34. For the rest of the hyperparameters used within CIFAR10 experiments (i.e., those used during pre-training), we adopt the settings of a widely-used, open-source repository that achieves good performance on CIFAR10 [160].

C i-SpaSP: Structured Neural Pruning via Sparse Signal Recovery

C.1 Experimental Details

C.1.1 Synthetic Experiments

Here, we present the details for generating the synthetic data used for pruning experiments with i-SpaSP on two-layer neural networks presented in Section 4.6.1. It should be noted that the synthetic data generated within this set of experiments does not correspond to the input data matrix $X \in \mathbb{R}^{d_{in} \times B}$ described in Section 4.2. Rather, the synthetic data that is generated $\{H^{(i)}\}_{i=1}^{K}$ corresponds to the hidden representation of a

[41]We adopt the same experimental settings, but decrease the number of fine-tuning epochs from 150 to 80 because we find that testing accuracy reaches a plateau well-before 150 epochs.

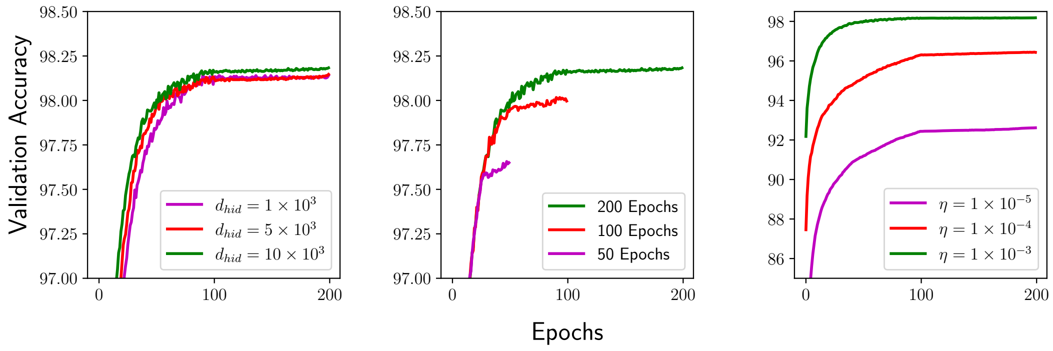

Figure 28: Performance of two-layer networks with different hyperparameter settings on the MNIST validation set. From left to right, subplots depict different hidden dimensions, pre-training epochs, and learning rates. For each of the plots, the best setting with respect to other hyperparameters is displayed.

two-layer neural network, constructed as $H^{(i)} = \sigma(W^{(0)} \cdot X)$ for some input X. Within these experiments, we generate hidden representations instead of raw input because the row-compressibility ratio p considered within Theorem 4 is with respect to the neural network's hidden representations (i.e., not with respect to the input).

Consider the i-th synthetic hidden representation matrix $H^{(i)} \in \mathbb{R}^{d_{hid} \times B}$, and assume this matrix has a desired row-compressibility ratio p. For all experiments, we set B (i.e., the size of the dataset) to be 100. Recall that all entries within $H^{(i)}$ must be non-negative due to the ReLU activation present within (16). For each row $j \in [d_{hid}]$ of the matrix, the synthetic hidden representation is generated by $i)$ computing the upper bound on the sum of the values within this j-th row as $\frac{R}{i^{\frac{1}{p}}}$ and $ii)$ randomly sampling B non-negative values that, when summed together, do not exceed the upper bound.[42] Here, R is a scalar constant as described in Section 4.2, which we set to $R = 1$. After this process has been completed for each row, the rows of $H^{(i)}$ are randomly shuffled so as to avoid rows being in sorted, magnitude order.

Within the experiments presented in Section 4.6.1, we generate three random matrices, following the process described above, for each combination of d_{hid} and p. Then, the average pruning results across each of these three matrices is reported for each experimental setting. Experiments were also replicated with different settings of R, but the results observed were quite similar. Further, the $W^{(1)}$ matrix used within the synthetic experiments of Section 4.6.1 was generated using standard Kaiming initialization [106], which is supported in deep learning packages like PyTorch [193].

C.1.2 Two-Layer Networks

Within this section, we provide all relevant experimental details for pruning experiments with two-layer networks presented in Section 4.6.2.

Baseline Network. We begin by describing the baseline two-layer network that was used within pruning

[42]In practice, this is implemented by keeping a running sum and continually sampling numbers randomly within a range that does not exceed the upper bound.

experiments on MNIST, as well as relevant details for pre-training the network prior to pruning. The network used within experiments in Section C.1.2 exactly matches the formulation in (16), and MNIST images are flattened—forming a 784-dimensional input vector—prior to being passed as input to the network. The network is first pre-trained on the MNIST dataset such that it has fully converged before being used in pruning experiments. During pre-training, we optimize the network with stochastic gradient descent (SGD) using a batch size of 128 and employ a learning rate step schedule that decays the learning rate $10\times$ after completing 50% and 75% of total epochs. For all network and pre-training hyperparameters, we identify optimal settings by dividing the MNIST training set randomly (i.e., a random, 80-20 split) into training and validation sets. Then, performance is measured over the validation set using three separate trials and averaged to identify proper hyperparameter settings.

We find that weight decay and momentum settings do not significantly impact network performance. Thus, we use no weight decay during pre-training and set momentum to 0.9. The results of other hyperparameter tuning experiments are provided in Figure 28. We test different hidden dimensions of the two-layer network $d_{hid} \in \{1 \times 10^3,\ 5 \times 10^3,\ 10 \times 10^3\}$, finding that validation performance reaches a plateau when $d_{hid} = 10 \times 10^3$. We also test numerous possible learning rates $\eta \in \{1 \times 10^{-5},\ 1 \times 10^{-4},\ 1 \times 10^{-3}\}$ and pre-training epochs. Within these experiments, it is determined that a learning rate of $\eta = 1 \times 10^{-3}$ with 200 epochs of pre-training acheives the best performance on the validation set; as shown in the right and middle subplots of Figure 28. Thus, the dense networks used for experiments within Section 4.6.2 has a hidden dimension of 10×10^3 and is pre-trained for 200 epochs using a learning rate of 1×10^{-3} prior to pruning being performed.

Network Pruning. The two-layer network described above is pruned using several greedy selection strategies, including i-SpaSP, GFS, and Top-K. Each of these pruning strategies selects a subset of neurons within the dense network's hidden layer based on a mini-batch of data from the training set. Within all experiments, we adopt a batch size of 512 for pruning.

For i-SpaSP, the pruning procedure follows the exact steps outlined in Algorithm 4. Within each iteration of Algorithm 4, a new mini-batch of data is sampled to compute neuron importance. We use a fixed number of iterations as the stopping criterion for i-SpaSP. In particular, we terminate the algorithm and return the pruned model after 20 iterations, as the active set of neurons is consistently stable at this point.

An in-depth description of GFS is provided in [251]. For two-layer networks, GFS is implemented by, at each iteration, sampling a new mini-batch of data and finding the neuron that, when included in the (initially empty) active set, yields the largest decrease in loss over the mini-batch. This process is repeated until the desired size of the pruned network has been reached—i.e., each iteration adds a single neuron to the pruned network and the size of the pruned network is used as a stopping criterion. Top-K is a naive, baseline method for greedy selection, which operates by computing hidden representations across the mini-batch and selecting the k neurons with the largest-magnitude hidden activations (i.e., based on summing hidden representation magnitudes across the mini-batch). Top-K performs the entire pruning process in one step using a single mini-batch.

C.1.3 Deep Convolutional Networks

Within this section, we provide relevant details to the large-scale CNN experiments on ImageNet (ILSVRC2012) presented in Section 4.6.3.

Pruning Deep Networks. As described in Section 4.4.1, i-SpaSP is extended to deep networks by greedily pruning each layer of the network from beginning to end. We prune filters within each block of the network independently, where a block either consists of two 3×3 convolution operations separated by batch normalization and ReLU (i.e., `BasicBlock` for ResNet34) or a depth-wise, separable convolution with a linear bottleneck (i.e., `InvertedResidualBlock` for MobileNetV2). We maintain the input and output channel resolution of each block—choosing instead to reduce each block's intermediate channel dimension—by performing pruning as follows:

- `BasicBlock`: We obtain the output of the first convolution (i.e., after batch normalization and ReLU) and perform pruning based on the forward pass of the second convolution, thus eliminating filters from the output of the first convolution and, in turn, the input of the second convolution.
- `InvertedResidualBlock`: We obtain the output of the first 1×1 point-wise and 3×3 depth-wise convolutions (i.e., after normalization and ReLU) and prune the filters of this representation based on the forward pass of the last 1×1 point-wise linear convolution, thus removing filters from the output of the first point-wise convolution and, in turn, within the depth-wise convolution operation.

Thus, each convolutional block is separated into two components. Input data is passed through the first component to generate a hidden representation, then i-SpaSP performs pruning using forward and backward passes of the second component as in Algorithm 4. Intuitively, one can view the two components of each block as the two neural network layers in (16). Due to implementation with automatic differentiation, pruning convolutional layers is nearly identical to pruning two-layer network as described in Section 4.4.1, the only difference being the need to sum over both batch and spatial dimensions in computing importance.

Pruning Ratios. As mentioned within Section 4.6.3, some blocks within each network are more sensitive to pruning than others. Thus, pruning all layers to a uniform ratio typically does not perform well. Within ResNet34, four groups of convolutional blocks exist, where each group contains a sequence of convolutional blocks with the same channel dimension. Following the recommendations of [162], we avoid pruning the fourth group of convolutional blocks, any strided blocks, or the last block in any group. The 2.69 GFlop model in Table 7 uses a uniform ratio of 40% for all three groups (i.e., 60% of filters are eliminated from each block). For the 2.13 GFlop model, the first group is pruned to a ratio of 20%, while the second and third groups are pruned to a ratio of 30%.

Within MobileNetV2, we simply avoid pruning network blocks that are either strided or have different input and output channel dimensions, leaving the following blocks to be pruned: 3, 5, 6, 8, 9, 10, 12, 13, 15, 16. We find that blocks 3, 5, 8, 15, and 16 are especially sensitive to pruning (i.e., pruning these layers causes noticeable performance degradation), so we avoid aggressive pruning upon these blocks. The 260 MFlop model is generated by pruning only blocks 6, 9, 10, 12, and 13 to a ratio of 40%. For the 242 MFlop and 220 MFlop models, we prune sensitive (i.e., blocks 3, 4, 8, 15, and 16) and not sensitive (i.e., blocks 6, 9, 10, 12, and 13) blocks with ratios of 40%/80% and 30%/70%, respectively. For the 220 MFlop model, we also prune blocks 2, 11, and 17 to a ratio of 90% to further decrease the FLOP count.

Pruning Hyperparameters. Pruning of each layer is performed with a batch of 1280 data examples for both ResNet34 and MobileNetV2 architectures. This batch of data can be separated into multiple mini-batches (e.g., of size 256) during pruning. Such an approach allows the computation of i-SpaSP to be performed on a GPU (i.e., all of the data examaples cannot simultaneously fit within memory of a typical GPU), but this modification does not change the runtime of behavior of i-SpaSP. Furthermore, 20 total i-SpaSP iterations are performed in pruning each layer. We find that adding more data or pruning iterations beyond these settings does not provide any benefit to i-SpaSP performance.

Fine-tuning Details. For fine-tuning, we perform 90 epochs of fine-tuning using SGD with momentum and employ a cosine learning rate decay schedule over the entire fine-tuning process beginning with a learning rate of 0.01. For fine-tuning between the pruning of layers (i.e., a small amount of fine-tuning is performed after each layer is pruned), we simply adopt the same fine-tuning setup with a fixed learning rate of 0.01 and fine-tuning continues for one epoch. Our learning rate setup for fine-tuning matches that of [251, 252], but we choose to perform fewer epochs of fine-tuning, as we find network performance does not improve much after 90 epochs. In all cases, we adopt standard augmentation procedures for the ImageNet dataset during fine-tuning (i.e., normalizing the data and performing random crops and flips).

C.1.4 Transformer Networks

Within this section, we provide all relevant experimental details for pruning experiments with two-layer networks presenting in Section 4.6.4.

Baseline Network. We begin by describing the baseline network used within the structured attention head pruning experiments. The network architecture used in all experiments is the multi-lingual BERT (mBERT) model [53]. More specifically, we adopt the BERT-base-cased variant of the architecture, which is pre-trained on a corpus of over 100 different languages. Beginning with this pre-trained network architecture, we fine-tune the BERT model on the XNLI dataset for two epochs with the AdamW optimizer and a learning rate of 5×10^{-5} that is decayed linearly throughout the training process. After the fine-tuning process, this model achieves an accuracy of 71.02% on the XNLI test set.

Network Pruning. For all pruning experiments, we adopt a structured pruning approach that selectively removes entire attention heads from network layers. We adopt two different pruning ratios: 25% and 40%.[43] These ratios correspond to three and five attention heads (i.e., each layer of mBERT originally has 12 attention heads) being retained within each layer of the network.

We compare the performance of i-SpaSP to a uniform pruning approach (i.e., randomly select attention heads to prune within each layer) and a sensitivity-based, global pruning approach that was previously proposed for structured pruning of attention heads [177]. For i-SpaSP and uniform pruning, one layer is pruned at a time, starting from the first layer of the network and ending at the final layer. For the sensitivity-based pruning technique, the pruning process is global and all layers are pruned simultaneously. After pruning

[43]Here, the pruning ratio refers to the number of attention heads remaining after pruning. For example, a 12-head layer pruned at a ratio of 25% would have three remaining heads after pruning completes.

has been completed, networks are fine-tuned for 0.5 epochs using the AdamW optimizer with a learning rate of 5×10^{-5} that is decayed linearly throughout the fine-tuning process. For i-SpaSP and the sensitivity-based pruning approach, we adopt the same setting as previous experiments (i.e., see Section 4.6.3) and utilize a subset of 1280 training data examples for pruning. Additionally, i-SpaSP performs 20 total iterations for the pruning of each layer, which again reflects the settings used within previous experiments.

C.2 Main Proofs

C.2.1 Gradient of $\mathcal{L}(\cdot,\cdot)$

Consider the objective function given in (17):

$$\begin{aligned}\mathcal{L}(U,U') &= \frac{1}{2}\|U - U'\|_F^2 \\ &= \frac{1}{2}\|W^{(1)}H - U'\|_F^2\end{aligned}$$

We now show that, considering U' as a constant matrix, $\nabla_H \mathcal{L}(U,U') = (W^{(1)})^\top \cdot V$.

Proof. Here, we consider U', the output of the pruned network, to be a constant matrix. To begin, consider the derivative of $\mathcal{L}$ with respect to a single element of the matrix H, given by the row index k and column index z.

$$\begin{aligned}\nabla_{H_{kz}} \frac{1}{2}\|U - U'\|_F^2 &= \nabla_{H_{kz}} \frac{1}{2}\sum_{ij}(U_{ij} - U'_{ij})^2 \\ &= \nabla_{H_{kz}} \sum_{ij} \frac{1}{2}(U_{ij} - U'_{ij})^2 \\ &\overset{i}{=} \sum_{ij}(U_{ij} - U'_{ij})\left(\nabla_{H_{kz}}(U_{ij} - U'_{ij})\right) \\ &\overset{ii}{=} \sum_{ij}(U_{ij} - U'_{ij})\left(\nabla_{H_{kz}} U_{ij}\right) \\ &= \sum_{ij}(U_{ij} - U'_{ij})\left(\nabla_{H_{kz}} W^{(1)}_{i,:} H_{:,j}\right) \\ &\overset{iii}{=} \sum_{i}(U_{iz} - U'_{iz})(W^{(1)}_{ik}) \\ &= \left(W^{(1)}_{:,k}\right)^\top \cdot (U_{:,z} - U'_{:,z})\end{aligned}$$

where i holds due to the chain rule, ii holds because entries of U' are considered constant, and iii holds by eliminating all elements from the sum where $\nabla_{H_{kz}} W^{(1)}_{i,:} H_{:,j} = 0$. From here, we can now aggregate all the derivatives $\nabla_{H_{kz}}\mathcal{L}(U, U')$ into a single Jacobian matrix as follows to arrive at the desired expression.

$$\begin{aligned}\nabla_H \mathcal{L}(U, U') &= \left(W^{(1)}\right)^\top \cdot \left(W^{(1)}H - U'\right) \\ &= \left(W^{(1)}\right)^\top \cdot V\end{aligned}$$

□

C.2.2 Properties of $\mu(\cdot)$

Prior to providing any proofs of our theoretical results, we show a few properties of the function $\mu(\cdot)$ defined over matrices that will become useful in deriving later results. Given $A \in \mathbb{R}^{m\times n}$, $\mu(A) : \mathbb{R}^{m\times n} \longrightarrow \mathbb{R}^m$ and is defined as follows:

$$\mu(A) = \sum_{i=1}^{n} A_{:,i}$$

In words, $\mu(A)$ sums all columns within an arbitrary matrix A to produce a single column vector. We now provide a few useful properties of $\mu(\cdot)$.

Lemma 9. *Consider two arbitrary matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$. The following properties of $\mu(\cdot)$ must be true.*

$$\mu(A \cdot B) = A \cdot \mu(B)$$
$$\mu(A + B) = \mu(A) + \mu(B)$$

Proof. Given the matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, we can arrive at the first property as follows:

$$\mu(A \cdot B) = \sum_{i=1}^{p}(A \cdot B)_{:,i} = \sum_{i=1}^{p} A \cdot B_{:,i} = A \cdot \left(\sum_{i=1}^{p} B_{:,i}\right) = A \cdot \mu(B)$$

Furthermore, the second property can be shown as follows:

$$\mu(A + B) = \sum_{i=1}^{p}(A + B)_{:,i} = \sum_{i=1}^{p} A_{:,i} + B_{:,i} = \sum_{i=1}^{p} A_{:,i} + \sum_{j=1}^{p} B_{:,j} = \mu(A) + \mu(B)$$

□

C.2.3 Proof of Lemma 3

Here, we provide a proof for Lemma 3 from Section 4.5.

Proof. Consider an arbitrary iteration of Algorithm 4. We define the active neurons within the pruned network entering this iteration as $\mathcal{S}$. The notation $\mathcal{S}'$ will be used to refer to the active neurons in the pruned model after the completion of the iteration. Similarly, we will use the $\cdot'$ notation to refer to other constructions after the iteration completes (e.g., V becomes V', R becomes R', etc.)

We begin by defining and expanding the definitions of the matrices U, V, R, and Y considering the substitution of $H = Z + E$;

$$\begin{aligned}
&U = W^{(1)}H = W^{(1)}(Z + E),\\
&V = U - W^{(1)}H_{\mathcal{S},:} = W^{(1)}(Z + E) - W^{(1)}(Z + E)_{\mathcal{S},:},\\
&R = Z - Z_{\mathcal{S},:},\\
&Y = (W^{(1)})^T V.
\end{aligned}$$

Now, observe the following about Y.

$$\begin{aligned}
Y &= (W^{(1)})^\top \cdot V\\
&= (W^{(1)})^\top \left(W^{(1)}(Z + E) - W^{(1)}(Z + E)_{\mathcal{S},:}\right)\\
&= (W^{(1)})^\top \left(W^{(1)}Z + W^{(1)}E - W^{(1)}(Z_{\mathcal{S},:}) - W^{(1)}(E_{\mathcal{S},:})\right)\\
&= (W^{(1)})^\top \left(W^{(1)}(Z - Z_{\mathcal{S},:}) + W^{(1)}(E - E_{\mathcal{S},:})\right)\\
&= (W^{(1)})^\top \left(W^{(1)}(Z - Z_{\mathcal{S},:}) + W^{(1)}\hat{E}\right)\\
&= (W^{(1)})^\top W^{(1)}(Z - Z_{\mathcal{S},:}) + (W^{(1)})^\top W^{(1)}\hat{E},
\end{aligned}$$

where we denote $\hat{E} = E - E_{\mathcal{S},:}$. It should be noted that $\hat{E}$ is simply the matrix E with all the i-th rows set to zero where $i \in \mathcal{S}$. Invoking Lemma 9 in combination with the above expression, we can show the following.

$$\begin{aligned}
\mu(Y) &= (W^{(1)})^\top W^{(1)}\mu(Z - Z_{\mathcal{S},:}) + (W^{(1)})^\top W^{(1)}\mu(\hat{E})\\
&= (W^{(1)})^\top W^{(1)} \left(\mu(R) + \mu(\hat{E})\right).
\end{aligned}$$

Based on the definition of $H = Z + E$, R must be s-row-sparse. This is because Z is s-row-sparse and $R = Z - Z_{\mathcal{S},:}$. Thus, the number of non-zero rows within R must be less than or equal to s (i.e., the rows that are not subtracted out of Z by $Z_{\mathcal{S},:}$). Denote $\Delta = \mathrm{rsupp}(R)$, where $|\Delta| \leq s$. From the definition of Ω in Algorithm 4, we have the following properties of Y:

$$\|\mu(Y)|_{\Delta}\|_2 \leq \|\mu(Y)|_{\Omega}\|_2 \quad \overset{i)}{\Longrightarrow} \quad \|\mu(Y)|_{\Delta\setminus\Omega}\|_2 \leq \|\mu(Y)|_{\Omega\setminus\Delta}\|_2$$

where $i)$ follows by removing indices within $\Omega \cap \Delta$. From here, we can upper bound the value of $\|\mu(Y)|_{\Omega\setminus\Delta}\|_2$

as follows.

$$\begin{aligned}
\|\mu(Y)|_{\Omega\setminus\Delta}\|_2 &= \left\| \left((W^{(1)})^\top W^{(1)} \mu(R) \right) |_{\Omega\setminus\Delta} + \left((W^{(1)})^\top W^{(1)} \mu(\hat{E}) \right) |_{\Omega\setminus\Delta} \right\|_2 \\
&= \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(R) + (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{i)}{\leq} \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(R) \right\|_2 + \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{ii)}{\leq} \delta_{3s} \|\mu(R)\|_2 + \left\| (W^{(1)}_{:,\Omega\setminus\Delta})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{iii)}{\leq} \delta_{3s} \|\mu(R)\|_2 + \sqrt{1+\delta_{2s}} \|W^{(1)} \mu(\hat{E})\|_2 \\
&\overset{iv)}{\leq} \delta_{3s} \|\mu(R)\|_2 + (1+\delta_{2s}) \|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}} \|\mu(\hat{E})\|_1
\end{aligned}$$

where $i)$ follows from the triangle inequality, $ii)$ holds from Corrolary 3.3 on RIP properties of $W^{(1)}$ in [184] because $|\Omega \cup \Delta| \leq 3s$, $iii)$ holds from Proposition 3.1 on RIP properties of $W^{(1)}$ in [184] because $|\Omega \setminus \Delta| \leq 2s$, and $iv)$ holds from Proposition 3.5 on RIP properties of $W^{(1)}$ in [184]. Now, we can also lower bound the value of $\|\mu(Y)|_{\Delta\setminus\Omega}\|_2$ as follows:

$$\begin{aligned}
\|\mu(Y)|_{\Delta\setminus\Omega}\|_2 &= \left\| \left((W^{(1)})^\top W^{(1)} \mu(R) \right) |_{\Delta\setminus\Omega} + \left((W^{(1)})^\top W^{(1)} \mu(\hat{E}) \right) |_{\Delta\setminus\Omega} \right\|_2 \\
&= \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R) + (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&= \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Delta\setminus\Omega} + (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Omega} + (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{i)}{\geq} \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Delta\setminus\Omega} \right\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Omega} \right\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{ii)}{\geq} (1-\delta_s) \|\mu(R)|_{\Delta\setminus\Omega}\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(R)|_{\Omega} \right\|_2 - \left\| (W^{(1)}_{:,\Delta\setminus\Omega})^\top W^{(1)} \mu(\hat{E}) \right\|_2 \\
&\overset{iii}{\geq} (1-\delta_s) \|\mu(R)|_{\Delta\setminus\Omega}\|_2 - \delta_s \|\mu(R)|_{\Omega}\|_2 - \sqrt{1+\delta_s} \|W^{(1)} \mu(\hat{E})\|_2 \\
&\overset{iv)}{\geq} (1-\delta_s) \|\mu(R)|_{\Delta\setminus\Omega}\|_2 - \delta_s \|\mu(R)\|_2 - (1+\delta_{2s}) \|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}} \|\mu(\hat{E})\|_1
\end{aligned}$$

where $i)$ follows from the triangle inequality. $ii)$ follows from Proposition 3.1 in [184] and $iii)$ follows from Corollary 3.3 and Proposition 3.1 in [184] because $|\Delta \setminus \Omega| \leq s$. Finally, $iv)$ follows from Proposition 3.5 within [184]. By noting that $\mu(R)|_{\Delta\setminus\Omega} = \mu(R)|_{\Omega^c}$ and combining the lower and upper bounds above, we derive the following:

$$\|\mu(R)|_{\Omega^c}\|_2 \leq \frac{(\delta_s + \delta_{3s}) \|\mu(R)\|_2 + 2 \left((1+\delta_{2s}) \|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}} \|\mu(\hat{E})\|_1 \right)}{1-\delta_s} \tag{43}$$

From here, we can show the following:

$$\begin{aligned}
\|\mu(R)|_{\Omega^c}\|_2 &\overset{i)}{\geq} \|\mu(R)|_{\Omega^{\star c}}\|_2 \\
&= \|\mu(Z - Z_{\mathcal{S},:})|_{\Omega^{\star c}}\|_2 \\
&= \|(\mu(Z) - \mu(Z)|_{\mathcal{S}})|_{\Omega^{\star c}}\|_2 \\
&\overset{ii)}{=} \|\mu(Z)|_{\Omega^{\star c}}\|_2 \\
&= \|\mu(H - E)|_{\Omega^{\star c}}\|_2 \\
&= \|\mu(H)|_{\Omega^{\star c}} - \mu(E)|_{\Omega^{\star c}}\|_2 \\
&\overset{iii)}{\geq} \|\mu(H)|_{\Omega^{\star c}}\|_2 - \|\mu(E)|_{\Omega^{\star c}}\|_2
\end{aligned} \tag{44}$$

where $i)$ follows from the fact that $\Omega \subseteq \Omega^\star$, $ii)$ follows from the fact that $\mathcal{S} \subset \Omega^\star$, and $iii)$ follows from the triangle inequality. Then, as a final step, we make the following observation, where we leverage notation from Algorithm 4:

$$\begin{aligned}
\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 &= \|\mu(H) - b_s\|_2 \\
&= \|(\mu(H) - b) + (b - b_s)\|_2 \\
&\overset{i)}{\leq} \|\mu(H) - b\|_2 + \|b - b_s\|_2 \\
&\overset{ii)}{\leq} 2\|\mu(H) - b\|_2 \\
&= 2\|\mu(H) - \mu(H)|_{\Omega^\star}\|_2
\end{aligned} \tag{45}$$

where $i)$ follow from the triangle inequality and $ii)$ follows from the fact that b_s is the best s-sparse approximation to b (i.e., $\mu(H)$ is a worse s-sparse approximation with respect to the ℓ_2 norm). Now, noticing that $\|\mu(H) - \mu(H)|_{\Omega^\star}\|_2 = \|\mu(H)|_{\Omega^{\star c}}\|_2$, we can aggregate all of this information as follows:

$$\begin{aligned}
\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 - 2\|\mu(\hat{E})\|_2 &\overset{i)}{\leq} \|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 - 2\|\mu(E)|_{\Omega^{\star c}}\|_2 \\
&\overset{ii)}{\leq} 2\left(\|\mu(H) - \mu(H)|_{\Omega^\star}\|_2 - \|\mu(E)|_{\Omega^{\star c}}\|_2\right) \\
&= 2\left(\|\mu(H)|_{\Omega^{\star c}}\|_2 - \|\mu(E)|_{\Omega^{\star c}}\|_2\right) \\
&\overset{iii)}{\leq} 2\|\mu(R)|_{\Omega^c}\|_2 \\
&\overset{iv)}{\leq} \frac{2(\delta_s + \delta_{3s})\|\mu(R)\|_2 + 4\left((1 + \delta_{2s})\|\mu(\hat{E})\|_2 + \frac{1+\delta_{2s}}{\sqrt{2s}}\|\mu(\hat{E})\|_1\right)}{1 - \delta_s}
\end{aligned}$$

where $i)$ holds because $\mathcal{S} \subset \Omega^\star$, $ii)$ holds from (45), $iii)$ holds from (44), and $iv)$ hold from (43). By simply moving the $\|\mu(\hat{E})\|_2$ term to the other side of the inequality, we get the following:

$$\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 \leq \frac{2(\delta_s + \delta_{3s})\|\mu(R)\|_2 + (6 + 4\delta_{2s} - 2\delta_s)\|\mu(\hat{E})\|_2 + \frac{4(1+\delta_{2s})}{\sqrt{2s}}\|\mu(\hat{E})\|_1}{1 - \delta_s}$$

From here, we recover $\mu(H) - \mu(H)|_{\mathcal{S}}$ from $\mu(R)$ as follows:

$$\begin{aligned}
\|\mu(R)\|_2 &= \|\mu(Z - Z|_{\mathcal{S}})\|_2 \\
&= \|\mu(H - E - (H - E)|_{\mathcal{S}})\|_2 \\
&= \|\mu(H - H|_{\mathcal{S}}) - \mu(E - E|_{\mathcal{S}})\|_2 \\
&\overset{i)}{\leq} \|\mu(H - H|_{\mathcal{S}})\|_2 + \|\mu(E - E|_{\mathcal{S}})\|_2 \\
&\leq \|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + \|\mu(\hat{E})\|_2
\end{aligned}$$

where $i)$ holds from the triangle inequality. Combining this with the final inequality derived above, we arrive at the following recursion for error between pruned and dense model hidden layers over iterations of Algorithm 4:

$$\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 \leq \frac{2(\delta_s+\delta_{3s})\|\mu(H)-\mu(H)|_{\mathcal{S}}\|_2+(6+2\delta_{3s}+4\delta_{2s})\|\mu(\hat{E})\|_2+\frac{4(1+\delta_{2s})}{\sqrt{2s}}\|\mu(\hat{E})\|_1}{1-\delta_s}$$

We now adopt an identical numerical assumption as [184] and assume that $W^{(1)}$ has restricted isometry constant $\delta_{4s} \leq 0.1$. Then, noting that $\delta_s, \delta_{2s}, \delta_{3s} \leq \delta_{4s} \leq 0.1$ per Corollary 3.4 in [184], we substitute the restricted isometry constants into the above recursion to yield the following expression:

$$\begin{aligned}
\|\mu(H) - \mu(H)|_{\mathcal{S}'}\|_2 &\leq 0.444\|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + 7.333\|\mu(\hat{E})\|_2 + \frac{4.888}{\sqrt{2s}}\|\mu(\hat{E})\|_1 \\
&= 0.444\|\mu(H) - \mu(H)|_{\mathcal{S}}\|_2 + 7.333\|\mu(\hat{E})\|_2 + \frac{3.456}{\sqrt{s}}\|\mu(\hat{E})\|_1
\end{aligned}$$

If we invoke Lemma 9, unroll the above recursion over t iterations of Algorithm 4, notice it is always true that $\|\hat{E}\|_2 \leq \|E\|_2$ and $\|\hat{E}\|_1 \leq \|E\|_1$, and draw upon the property of ℓ_1 and ℓ_2 vector norms that $\|\cdot\|_2 \leq \|\cdot\|_1$ we arrive at the desired result:

$$\begin{aligned}
\|\mu(H - H_{\mathcal{S}_t,:})\|_2 &= \|\mu(H) - \mu(H)|_{\mathcal{S}_t}\|_2 \\
&\leq (0.444)^t\|\mu(H) - \mu(H)|_{\mathcal{S}_0}\|_2 + 14\|\mu(E)\|_2 + \frac{7}{\sqrt{s}}\|\mu(E)\|_1 \\
&\leq (0.444)^t\|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right)\|\mu(E)\|_1
\end{aligned}$$

□

C.2.4 Proof of Theorem 3

Here, we provide a proof of Theorem 3 from Section 4.5.

Proof. Consider an arbitrary iteration t of Algorithm 4 with a set of active neurons within the pruned model denoted by $\mathcal{S}_t$. The matrix V_t contains the residual between the dense and pruned model output over the entire dataset, as formalized below:

$$\begin{aligned} V_t &= U - (W^{(1)} \cdot H_{\mathcal{S}_t,:}) \\ &= (W^{(1)} \cdot H) - (W^{(1)} \cdot H_{\mathcal{S}_t,:}) \\ &= W^{(1)} \cdot (H - H_{\mathcal{S}_t,:}) \end{aligned}$$

Consider the squared Frobenius norm of the matrix V_t. We can show the following:

$$\begin{aligned} \|V_t\|_F^2 &= \|W^{(1)} \cdot (H - H_{\mathcal{S}_t,:})\|_F^2 \\ &= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(\sum_{z=1}^{d_{hid}} W_{iz}^{(1)} (H - H_{\mathcal{S}_t,:})_{zj} \right)^2 \\ &= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(W_{i,:}^{(1)} \cdot (H - H_{\mathcal{S}_t,:})_{:,j} \right)^2 \\ &= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(|W_{i,:}^{(1)} \cdot (H - H_{\mathcal{S}_t,:})_{:,j}| \right)^2 \\ &\overset{i)}{\leq} \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(\|W_{i,:}^{(1)}\|_2 \cdot \|(H - H_{\mathcal{S}_t,:})_{:,j}\|_2 \right)^2 \\ &= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \|W_{i,:}^{(1)}\|_2^2 \cdot \|(H - H_{\mathcal{S}_t,:})_{:,j}\|_2^2 \\ &= \sum_{i=1}^{d_{out}} \sum_{j=1}^{B} \left(\sum_{k=1}^{d_{hid}} (W_{ik}^{(1)})^2 \right) \left(\sum_{z=1}^{d_{hid}} (H - H_{\mathcal{S}_t,:})_{zj}^2 \right) \\ &= \left(\sum_{i=1}^{d_{out}} \sum_{k=1}^{d_{hid}} (W_{ik}^{(1)})^2 \right) \left(\sum_{j=1}^{B} \sum_{z=1}^{d_{hid}} (H - H_{\mathcal{S}_t,:})_{zj}^2 \right) \\ &= \|W^{(1)}\|_F^2 \left(\sum_{z=1}^{d_{hid}} \sum_{j=1}^{B} (H - H_{\mathcal{S}_t}, :)_{zj}^2 \right) \\ &\overset{ii)}{\leq} \|W^{(1)}\|_F^2 \left(\sum_{z=1}^{d_{hid}} \mu(H - H_{\mathcal{S}_t,:})_z^2 \right) \\ &= \|W^{(1)}\|_F^2 \cdot \|\mu(H - H_{\mathcal{S}_t,:})\|_2^2 \end{aligned}$$

where $i)$ follows from the Cauchy-Schwarz inequality and $ii)$ follows from the fact that all entries of $H - H_{\mathcal{S}_t,:}$ are non-negative. From here, we simply invoke Lemma 3 to arrive at the desired result.

$$\begin{aligned}\|V_t\|_F &\leq \|W^{(1)}\|_F \cdot \|\mu(H - H_{\mathcal{S}_t,:})\|_2 \\ &\leq \|W^{(1)}\|_F \cdot \left((0.444)^t \|\mu(H)\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right)\|\mu(E)\|_1\right)\end{aligned}$$

□

C.2.5 Proof of Lemma 4

Here, we provide the proof for Lemma 4 from Section 4.5.

Proof. Assume that we choose the s-row-sparse Z matrix within $H = Z + E$ as follows

$$Z = H_{\mathcal{D},:}, \text{ where } \mathcal{D} = \underset{\mathcal{D}\subset[d_{hid}]}{\arg\min} \|\mu(H) - \mu(H)|_{\mathcal{D}}\|$$

The norm used within the above expression is arbitrary, as we simply care to select indices $\mathcal{D}$ such that $\mu(H)|_{\mathcal{D}}$ contains the largest-magnitude components of $\mu(H)$. Such a property will be true for any ℓ_p norm selection within the above equation, as they all ensure $\mu(H)|_{\mathcal{D}}$ is the best possible s-sparse approximation of $\mu(H)$. From here, we have that $E = H - H_{\mathcal{D}}$, which yields the following:

$$\begin{aligned}\|\mu(E)\|_1 &= \|\mu(H - H_{\mathcal{D}})\|_1 \\ &\overset{i)}{=} \|\mu(H) - \mu(H)|_{\mathcal{D}}\|_1 \\ &\overset{ii)}{\leq} R \cdot \frac{s^{1-\frac{1}{p}}}{\frac{1}{p} - 1}\end{aligned}$$

where $i)$ holds from Lemma 9 and $ii)$ is a bound on the ℓ_1 norm of the best-possible s-sparse approximation of a p-compressible vector; see Section 2.6 in [184].

□

C.2.6 Proof of Theorem 5

Here, we generalize our theoretical results beyond single-hidden-layer networks to arbitrary depth networks. For this section, we define the forward pass in a L-hidden-layer network via the following recursion:

$$H^{(\ell)} = \sigma(W^{(\ell-1)} \cdot H^{(\ell-1)})$$

where σ denotes the nonlinear ReLU activation applied at each layer and we have weight matrices $\mathcal{W} = \{W^{(0)}, W^{(1)}, \dots, W^{(L)}\}$ and hidden representations $\mathcal{H} = \{H^{(0)}, H^{(1)}, \dots, H^{(L)}\}$ for each of the L layers within the network. Here, $H^{(L)}$ denotes the network output and $H^{(0)} = \sigma(W^{(0)} \cdot X)$. It should be noted that no ReLU activation is applied at the final network output (i.e., $H^{(L)} = W^{(L)} \cdot H^{(L-1)}$). We denote the pre-activation values each hidden representation as $U^{(\ell)} = W^{(\ell)} \cdot H^{(\ell-1)}$.

Proof. We denote the active set of neurons after t iterations at each layer ℓ as $\mathcal{S}_t^{(\ell)}$. The dense network output is given by $U^{(L)} = H^{(L)}$, and we compute the residual between pruned and dense networks at layer ℓ as $V_t^{(\ell)} = U^{(\ell)} - W^{(\ell)}_{:,\mathcal{S}_t^{(\ell)}} \cdot H^{(\ell-1)}_{\mathcal{S}_t^{(\ell)},:}$. We further denote $U_t^{(\ell)'} = W^{(\ell)}_{:,\mathcal{S}_t^{(\ell)}} \cdot H^{(\ell-1)}_{\mathcal{S}_t^{(\ell)},:}$ as shorthand for intermediate, pre-activation outputs of the pruned network. For simplicity, we assume all hidden layers of the multi-layer network have the same number of hidden neurons d and are pruned to a size s, such that $s \ll d$. Consider the final output representation of an L-hidden-layer network $H^{(L)}$. We will now bound $\|V_t^{(L)}\|_F$, revealing that a recursion can be derived over the layers of the network to upper bound the norm of the final layer's residual between pruned and dense networks.

Following previous proofs, we decompose the hidden representation at each layer prior to the output layer as $H^{(\ell)} = Z^{(\ell)} + E^{(\ell)} + \Delta^{(\ell)}$, where $Z^{(\ell)}$ is an s-row-sparse matrix and $E^{(\ell)}$ and $\Delta^{(\ell)}$ are both arbitrary-valued matrices. $E^{(\ell)}$ denotes the same arbitrarily-valued matrix from the previous $H = Z + E$ formulation and $\Delta^{(\ell)}$ captures all error introduced by pruning layers prior to ℓ within the network. $E^{(\ell)}$ and $\Delta^{(L)}$ can also be combined into a single matrix as $\Xi^{(\ell)} = E^{(\ell)} + \Delta^{(\ell)}$. Now, we consider the value of $\|V_t^{(L)}\|_F$:

$$\begin{aligned}
\|V_t^{(L)}\|_F &\overset{i}{=} \|W^{(L)}\|_F \cdot \left((0.444)^t \|\mu(H^{(L-1)})\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(\Xi^{(L-1)})\|_1 \right) \\
&= \|W^{(L)}\|_F \cdot \left((0.444)^t \|\mu(H^{(L-1)})\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \|\mu(E^{(L-1)}) + \mu(\Delta^{(L-1)})\|_1 \right) \\
&\overset{ii}{\leq} \|W^{(L)}\|_F \cdot \left((0.444)^t \|\mu(H^{(L-1)})\|_2 + \left(14 + \frac{7}{\sqrt{s}}\right) \left(\|\mu(E^{(L-1)})\|_1 + \|\mu(\Delta^{(L-1)})\|_1 \right) \right) \\
&\overset{iii}{\approx} \|W^{(L)}\|_F \cdot \left(14 + \frac{7}{\sqrt{s}}\right) \left(\|\mu(E^{(L-1)})\|_1 + \|\mu(\Delta^{(L-1)})\|_1 \right) \qquad (46)
\end{aligned}$$

where i holds from Theorem 3, ii holds from the triangle inequality, and iii holds by eliminating all terms that approach zero with enough pruning iterations t. Now, notice that $\|\mu(\Delta^{(L-1)})\|_1$ characterizes the error induced by pruning within the previous layer of the network, following the ReLU activation. Thus, our expression for the error induced by pruning on network output is now expressed with respect to the pruning error of the previous layer, revealing that a recursion can be derived for pruning error over the entire multi-layer network. We now expand the expression for $\|\mu(\Delta^{(\ell)})\|_1$ (i.e., we use arbitrary layer ℓ to enable a recursion over layers to be derived).

$$
\begin{aligned}
\|\mu(\Delta^{(\ell)})\|_1 &\overset{i)}{=} \|\mu(\sigma(U^{(\ell)}) - \sigma(U_t^{(\ell)'}))\|_1 \\
&\overset{ii)}{\leq} \|\mu(\sigma(U^{(\ell)} - U_t^{(\ell)'}))\|_1 \\
&= \|\mu(\sigma(V_t^{(\ell)}))\|_1 \\
&= \left\|\mu\left(\sigma\left(W^{(\ell)} \cdot \left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)\right)\right)\right\|_1 \\
&\overset{iii)}{\leq} \sum_{i=1}^{d}\sum_{j=1}^{B}\left|\left(\sum_{z=1}^{d} W_{iz}^{(\ell)}\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{zj}\right)\right| \\
&= \sum_{i=1}^{d}\sum_{j=1}^{B}\left|W_{i,:}^{(\ell)} \cdot \left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{:,j}\right| \\
&\overset{iv)}{\leq} \sum_{i=1}^{d}\sum_{j=1}^{B}\|W_{i,:}^{(\ell)}\|_1\left\|\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{:,j}\right\|_1 \\
&= \sum_{i=1}^{d}\sum_{j=1}^{B}\left(\sum_{k=1}^{d}\left|W_{ik}^{(\ell)}\right|\right)\left(\sum_{z=1}^{d}\left|\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{zj}\right|\right) \\
&= \left(\sum_{i=1}^{d}\sum_{k=1}^{d}\left|W_{ik}^{(\ell)}\right|\right)\left(\sum_{z=1}^{d}\sum_{j=1}^{B}\left|\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)_{zj}\right|\right) \\
&= \left\|\mathtt{vec}(W^{(\ell)})\right\|_1 \cdot \left\|\mu\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)\right\|_1 \\
&\overset{v)}{\leq} \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1 \cdot \left\|\mu\left(H^{(\ell-1)} - H^{(\ell-1)}_{\mathcal{S}_t^{(\ell-1)},:}\right)\right\|_2 \\
&\overset{vi)}{\leq} \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1\left((0.444)^t\|\mu(H^{(\ell-1)})\|_2 + (14 + \frac{7}{\sqrt{s}})\|\mu(\Xi^{(\ell-1)})\|_1\right) \\
&\overset{vii)}{\leq} \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1\left((0.444)^t\|\mu(H^{(\ell-1)})\|_2 + (14 + \frac{7}{\sqrt{s}})\left(\|\mu(E^{(\ell-1)}\|_1 + \|\mu(\Delta^{(\ell-1)}\|_1\right)\right) \\
&\approx \sqrt{d}\left\|\mathtt{vec}(W^{(\ell)})\right\|_1(14 + \frac{7}{\sqrt{s}})\left(\|\mu(E^{(\ell-1)}\|_1 + \|\mu(\Delta^{(\ell-1)}\|_1\right)
\end{aligned}
$$

where $i)$ holds because $\Delta^{(\ell)}$ simply characterizes the difference between pruned and dense network hidden representations, $ii)$ holds due to properties of the ReLU function, $iii)$ hold by expanding the expression as a sum and replacing the ReLU operation with absolute value, $iv)$ holds due to the Cauchy Schwarz inequality, $v)$ holds due to properties of ℓ_1 norms, $vi)$ holds due to Lemma 3, and $vii)$ holds due to the triangle inequality. As can be seen within the above expression, the value of $\|\mu(\Delta^{(\ell)})\|_1$ can be expressed with respect to $\|\mu(\Delta^{(\ell-1)})\|_1$. Now, by beginning with (46) and unrolling the equation above over all L layers of the network, we obtain the following

$$\|V_t^{(L)}\|_F \leq \mathcal{O}\left(\sum_{i=1}^{L} d^{\frac{L-i}{2}} \left(14 + \frac{7}{\sqrt{s}}\right)^{L-i+1} \left\|\mu(E^{(i)})\right\|_1 \left(\|W^{(L)}\|_F \prod_{j=1}^{L-i} \|\texttt{vec}(W^{(j)})\|_1\right)\right)$$

Now, assume that $H^{(\ell)}$ is $p^{(\ell)}$-row-compressible with factor $R^{(\ell)}$ for $\ell \in [L-1]$. Then, defining $p = \max_i p^{(i)}$ and $R = \max_i R^{(i)}$, we invoke Lemma 4 to arrive at the desired result

$$\|V_t^{(L)}\|_F \leq \mathcal{O}\left(\sum_{i=1}^{L} \left(14 + \frac{7}{\sqrt{s}}\right)^{L-i+1} \left(\|W^{(L)}\|_F \prod_{j=1}^{L-i} \|\texttt{vec}(W^{(j)})\|_1\right) \left(\frac{d^{\frac{L-i}{2}} s^{1-\frac{1}{p}}}{\frac{1}{p} - 1}\right)\right)$$

where R is factored out because it has no asymptotic impact on the expression.

□

C.3 Supplementary Information

All code for this project is publicly-available via github at the following link: `https://github.com/wolfecameron/i-SpaSP`

D Cold Start Streaming Learning for Deep Networks

D.1 Experimental Details

D.1.1 Class-Incremental Learning

Here, we provide all details for the class-incremental streaming learning experiments presented in Section 5.6.1. For both CIFAR100 and ImageNet, the ordering of data (i.e., both by class and by example) is fixed for all methodologies, such that data is always encountered in the same order.

Offline Training Details. For CIFAR100, Top-1 Ω_{all} is computed using an offline-trained ResNet18 model that achieves a Top-1 accuracy of 78.61%. This model is trained using standard data augmentation for CIFAR100 (i.e., random crops and flips) using the SGD optimizer with momentum of 0.9 and an initial learning rate of 0.1. The learning rate is decayed throughout training using a cosine learning rate decay schedule over 200 total epochs. A weight decay of 5×10^{-4} is used. These offline training settings are adopted from a widely-used repository for achieving state-of-the-art performance on CIFAR10 and CIFAR100 [160].

Similarly, Top-5 Ω_{all} on ImageNet is computed using an offline-trained ResNet18 model that achieves a Top-5 accuracy of 89.09%. This pre-trained model is made publicly available via torchvision [193] and matches the offline normalization model used to evaluate class-incremental streaming learning on ImageNet in previous work [103].

Data Details. On CIFAR100, the dataset is divided into batches of 20 classes, where the first 20 classes are reserved for base initialization. Similarly, ImageNet is divided into batches of 100 classes, and the first 100 classes are used for base initialization. Evaluation occurs after each batch of data in the streaming process, and all testing events are aggregated within the Ω_{all} score. For all baselines, standard test augmentations for both CIFAR100 and ImageNet are adopted. Namely, because the feature extractors of ExStream, Deep SLDA, and REMIND are fixed, we perform appropriate resizing, center cropping, and normalization (i.e., following standard test settings of CIFAR100 and ImageNet) prior to passing each image into the fixed feature extractor. REMIND also leverages random resized crops and manifold Mixup [228] on the feature representations stored within the replay buffer throughout streaming. The proposed methodology leverages the data augmentation policy described in Section 5.3, and images are cropped and normalized following standard practice prior to augmentation.

Baseline Details. For ExStream, Deep SLDA, and REMIND, we utilize official, public implementations within all experiments [99, 100, 101]. ExStream is optimized using Adam with a learning rate of 2×10^{-3} and no weight decay, as in the official implementation. For CIFAR100, we modify the number of class exemplars (i.e., 10K, 20K, ..., 50K exemplars) to study ExStream's behavior with different replay buffer sizes. For ImageNet, we follow the settings of [103] and train ExStream using 20 and 100 exemplars per class for buffer capacities of 1,5Gb and 8Gb, respectively. REMIND uses the SGD with momentum optimizer with a learning rate that decays from 0.1 to 0.001 from the beginning to the end of each class. Momentum is set to 0.9 and weight decay to 1×10^{-5}. REMIND samples 100 and 50 replay samples during each online update for CIFAR100 and ImageNet, respectively. For Deep SLDA, we adopt the variant that does not fix the covariance matrix during the streaming process for all experiments. For REMIND, we perform experiments with a version that exactly matches the settings of the original papers, as well as with a version that adds two extra layers to the trainable portion of the ResNet18 model such that the number of trainable parameters between REMIND and CSSL is identical. All other experimental settings for each of the baseline methodologies exactly match the settings within each of the respective papers and/or public implementations [99, 100, 101].

All baseline methodologies perform base initialization over the first batch of data within the dataset. This base initialization includes 50 epochs of tine-tuning, followed by the initialization of any algorithm-specific modules to be used during the streaming process (e.g., the PQ module for REMIND). For fine-tuning, we adopt the same hyperparameter settings used for training the offline baseline models. After base initialization, streaming begins at the first class within the base initialization batch, meaning that data used within base initialization is re-visited during streaming. For ExStream and Deep SLDA, models are initialized with pre-trained ImageNet weights during base initialization on CIFAR100 experiments to enable more comparable experimental results on the smaller dataset.

CSSL Details. The proposed methodology is trained with a SGD optimizer with momentum of 0.9 and a learning rate that decays linearly from 0.1 to 0.001 from the first to last example of each class (i.e., the same learning rate decay strategy as [103] is adopted). We use weight decay of 1×10^{-5}. 100 replay samples are used within each online update for both CIFAR100 and ImageNet. We utilize the augmentation strategy described in Section 5.3 in all experiments. For Mixup and Cutmix, we utilize α values of 0.1 and 0.8, respectively. For AutoAugment, we adopt the CIFAR and ImageNet learned augmentation policies for each of the respective datasets. See Appendix D.2.1 for details regarding the derivation of the optimal augmentation policy and associated hyperparameters. For experiments that begin with a pre-trained parameter setting, we use identical training settings as used in base initialization for baseline methodologies to train model parameters over the first 20 classes of CIFAR100 or 100 classes of ImageNet, then use the resulting model parameters to initialize CSSL for streaming.

D.1.2 Different Data Orderings

Here, we overview the experimental details of the experiments performed using the Core50 dataset [164] in Section 5.6.2.

Data Orderings. Experiments are performed using four different data orderings: i.i.d. (ID), class i.i.d. (CID), instance (I), and class-instance (CI). Such data orderings are described in detail in [164], but we also provide a brief description of each ordering scheme below:

- `i.i.d.`: batches contain a uniformly random selection of images from the dataset.
- `class i.i.d.`: batches contain all images from two classes (i.e., out of ten total classes) in the dataset.
- `instance`: batches contain all images, in temporal order, corresponding to 80 unique object instances within the dataset.
- `class-instance`: batches contain images, in temporal order, corresponding to two classes in the dataset.

For each of the four possible ordering, ten unique data permutations are generated. Then, all experiments are repeated over each of these permutations, and performance is recorded for each ordering scheme as an average over all possible permutations.

Offline Training Details. Top-1 Ω_{all} on Core50 is computed using an offline-trained ResNet18 model that achieves a Top-1 accuracy of 43.91%. This model is trained for 40 epochs from a random initialization using SGD with momentum of 0.9 and an initial learning rate of 0.01. We use a weight decay of 1×10^{-4} and decay the learning rate by $10\times$ at epochs 15 and 30 during training. The settings of training our offline model for Core50 exactly match those provided in [103], aside from not initializing the model with pre-trained ImageNet weights. We attempted to improve the performance of the offline model by utilizing a greater number of epochs and modifying hyperparameter settings, but such tuning resulted in only negligible performance improvements.

Data Details. We sample the Core50 dataset at 1 frame per second, resulting in 600 and 225 training and testing images for each class [102]. The Core50 dataset has 10 total classes, resulting in 6000 total training images and 2250 total testing images. We adopt the same bounding box crops and splits from [164]. The dataset is split into batches of 1200 examples, where the content of each batch is determined by the ordering scheme and particular data permutation chosen for a particular experiment. Comparable experiments utilize both the same ordering scheme and the same permutation, where 10 unique permutations exist for each ordering scheme. The first batch is utilized for base initialization, and evaluation occurs after each batch to compute the final Ω_{all} score. To match the settings of [103], we do not utilize data augmentation within any of

Dataset	Classes	Training Examples	Testing Examples
CUB-200	200	5994	5794
Oxford Flowers	102	2040	6149
MIT Scenes	67	5360	1340
FGVC-Aircrafts	100	6667	3333

Table 22: Details of datasets used for multi-task streaming learning experiments.

the baseline methodologies (i.e., adding data augmentation was shown to degrade performance on Core50). The proposed methodology utilizes the same data augmentation policy that is described in Section 5.3.

Baseline Details. Again, we utilize official, public implementations of ExStream, Deep SLDA, and REMIND [102, 103, 104]. ExStream is optimized using Adam with a learning rate of 2×10^{-3} and no weight decay, and the memory buffer is allowed to store the full dataset (i.e., this can be done with < 100Mb of memory). REMIND is trained using SGD with momentum of 0.9 and a fixed learning rate of 0.01. A weight decay of 1×10^{-4} is used and 20 samples are taken during each online update, which matches settings of [103]. For Deep SLDA, we again adopt the variant that does not fix the covariance matrix during the streaming process. All other experimental settings match the hyperparameters provided in the respective papers and/or public implementations [99, 100, 101]. No data augmentation is used during baseline streaming experiments, as outline in [103].

The first batch of data is used for base initialization within all streaming baseline methodologies. During base initialization, the fine-tuning procedure again adopts the same hyperparameters as the offline training procedure described above. After the network has been fine-tuned over base intialization data, all algorithm-specific modules are initialized using the same data, and streaming begins from the first class of the base initialization step. We do not utilize pre-trained ImageNet weights to initialize baseline model parameters within Core50 experiments, as we are attempting to accurately assess model performance in low-resource scenarios on Core50 experiments.

CSSL Details. The proposed methodology is trained with SGD with momentum of 0.9 and a fixed learning rate of 0.01. We use a weight decay of 1×10^{-4}. 100 replay samples are observed during each online update, and we adopt the data augmentation policy described in Section 5.3. For AutoAugment, we utilize the ImageNet augmentation policy, and α values of 0.1 and 0.8 are again adopted for Mixup and CutMix, respectively.

The proposed methodology is tested with replay buffer capacities of 100, 200, and 300Mb. For the 200Mb experiment, image pixels are quantized to four bits, while for the 100Mb experiment we both quantize pixels and resize images to 75% of their original area. The 300Mb experiment performs no quantization or resizing, as the full dataset can be stored as raw images with memory overhead slightly below 300Mb.

D.1.3 Multi-Task Streaming Learning

Here, we overview the details of the multi-task streaming learning experiments in Section 5.6.3.

Datasets. We perform multi-task streaming learning with the CUB-200 [229], Oxford Flowers [188], MIT

Scenes [197], and FGVC-Aircrafts datasets [172]. The details of each of these datasets are provided in Table 22. Following the settings of [118], the datasets are learned in the following order: MIT Scenes, CUB-200, Oxford Flowers, then FGVC-Aircrafts.

Offline Training Details. The offline trained models for multi-task streaming experiments are trained separately for each of the datasets listed in Table 22. For each dataset, offline models are trained for 50 total epochs with cosine learning rate decay. Initial learning rates of 0.1, 0.01, and 0.001 are tested and results of the best-performing model are reported in Section 5.6.3. We also test baseline models with step learning rate schedules but find that cosine learning rate decay tends to produce models with better performance.

Data Details. For multi-task streaming learning performance, testing is performed after each dataset has been observed. We choose to not perform more regular evaluation (e.g., after subsets of each individual dataset have completed streaming) to make performance simple to interpret. All results are reported in terms of Top-1 accuracy computed separately on each dataset at the end of the streaming process. During streaming, each dataset is observed one example at a time in a class-incremental order. Three separate class-incremental data permutations are generated, and results are averaged across these three different permutations. The order of the datasets—presented above—is kept fixed in the three different permutations, as we only shuffle class and example orderings.

Because the feature extractors of ExStream, Deep SLDA, and REMIND are fixed, we perform appropriate resizing, center cropping, and normalization prior to passing each image into the fixed feature extractor. REMIND also leverages random resized crops and manifold mixup [228] on the feature representations stored within the replay buffer throughout streaming. The proposed methodology leverages the data augmentation policy described in Section 5.3, and images are cropped and normalized following standard practice prior to augmentation. Baselines utilize standard training and testing augmentations as used on ImageNet, which matches the settings of [118].

Baseline Details. Again, official, public implementations of baseline methodologies with the ResNet18 architecture are used in all experiments. For baseline methodologies, base initialization is performed over 50% of the MIT scenes dataset (i.e., the first dataset to be observed during streaming), encompassing 34 of the 67 total classes. All baseline methodologies are allowed unlimited memory capacity for replay. ExStream is optimized using Adam with a learning rate of 2×10^{-3} and no weight decay. REMIND uses the stochastic gradient descent (SGD) optimizer with a learning rate that decays from 0.1 to 0.001 from the beginning to the end of each class. Momentum is set to 0.9 and weight decay to 1×10^{-5}. REMIND samples 50 examples from the replay buffer for each online update. Furthermore, we modify REMIND's replay strategy to sample a batch of replayed data from each of the seen tasks when performing each online update. Such a change is necessary to not forget previous tasks when learning each new dataset, and we do not enforce any memory capacity on the replay buffer (i.e., all data can be kept for replay). For Deep SLDA, we adopt the variant that does not fix the covariance matrix during the streaming process for all experiments.

All baseline methodologies perform base initialization over the first 34 classes of the MIT scenes dataset. This base initialization includes 50 epochs of tine-tuning, followed by the initialization of any algorithm-specific modules to be used during the streaming process (e.g., the PQ module for REMIND). For fine-tuning, we adopt the same hyperparameter settings used for training the offline baseline models. After base initialization, streaming begins at the first class within the base initialization batch, meaning that data used within base initialization is re-visited during streaming. For each of the baseline methodologies, we perform experiments both with and without pre-training on ImageNet. Experiments with pre-training simply initialize the underlying ResNet18 architecture with ImageNet pre-trained weights prior to performing base initialization.

CSSL Details. The proposed methodology is trained with an SGD optimizer with momentum of 0.9 and a learning rate that decays linearly from 0.1 to 0.001 from the beginning to the end of each class. Weight decay of 1×10^{-5} is used. For all datasets, 100 replay samples are taken at each online update, and we modify the replay strategy to sample a batch of replayed data from each of the seen tasks when performing an online update. The augmentation strategy described in 5.3 is used in all experiments. Mixup and Cutmix use α values of 0.1 and 0.8, respectively, and AutoAugment adopts the Imagenet learned augmentation policy. See Appendix D.2.1 for details regarding the derivation of the optimal augmentation strategy. For all experiments, CSSL is fine-tuned on the first 34 classes of the MIT scenes dataset with identical hyperparameters as are used for base initialization with other methodologies. Some experiments utilize ImageNet pre-training and are identified as such.

D.1.4 Confidence Calibration Analysis

Here, we present experimental details for the confidence calibration analysis performed in Section 5.6.4.

Expected Calibration Error. Confidence calibration is measured in terms of expected calibration error (ECE). Assume that the i-th example within the testing set is associated with a true label, a model prediction, and a confidence value. Additionally, assume N total examples exist within the test set. Given N_{bin} bins, ECE groups data into uniformly-spaced bins based on confidence values. For example, if $N_{\text{bin}} = 2$, all predictions are separated into two groups, one group within confidence values in the range $[0.0, 0.5)$ and another with confidence values in the range $[0.5, 1.0]$.

Denote the i-th bin as bin_i. Once model predictions are separated into such bins, ECE computes the average accuracy and confidence of predictions within each bin. Then, using a_i and c_i to denote the accuracy and average confidence within the i-th bin, ECE can be computed as shown in the equation below.

$$\texttt{ECE} = \sum_{i=1}^{N_{\text{bin}}} \frac{|\text{bin}_i|}{N} \cdot |a_i - c_i|$$

For all experiments within Section 5.6.4, we follow the settings of [90] and set $N_{\text{bin}} = 15$. Performance is sometimes reported in terms of average ECE. In such cases, average ECE is computed by measuring ECE separately at each testing event during the streaming process, then taking a uniform average of ECE values across all testing events.

CIFAR100 Experiments. For the CIFAR100 experiments reported in Table 14, we perform class-incremental streaming using the same data ordering described in Appendix D.1.1. CSSL experiments use two different memory capacities: 30Mb (10K CIFAR100 images) and 150Mb (full CIFAR100 dataset). The version of

Model	Top-5 Acc.
ResNet101	93.54%
MobileNetV2	90.29%
DenseNet121	91.98%
Wide ResNet50-2	94.09%

Table 23: Top-5 accuracies achieved by offline models of different architectures on ImageNet.

CSSL denoted as "No Aug." replaces the CSSL augmentation strategy outlined in Section 5.3 with a simple augmentation policy that performs only random crops and flips. Furthermore, the pre-trained variant of CSSL simply initializes model parameters with pre-trained ImageNet weights at the beginning of the streaming process. All other experimental settings are identical to that of the class incremental learning experiments outlined in Appendix D.1.1.

For the streaming experiment depicted in Figure 13, we perform streaming with an i.i.d. ordering of the CIFAR100 dataset. Because the data ordering is i.i.d., all testing events perform evaluation over the entire CIFAR100 testing dataset. For CSSL, a memory capacity of 150Mb is used for the replay buffer. All experiments with CSSL utilize the augmentation strategy described in Section 5.3, except for the "No Aug." experiment that utilizes only random crops and flips for data augmentation. REMIND is allowed to store the full dataset for replay. All other experimental settings match those of the class-incremental learning experiments for CIFAR100 described in Appendix D.1.1.

ImageNet. For the confidence calibration experiments on ImageNet presented in Table 14, we utilize a class-incremental data ordering for streaming. For CSSL, we use a 1.5Gb replay buffer capacity and store all images with JPEG compression on disk. For the "No Aug." variant of CSSL, we again use random crops and flips for data augmentation instead of the augmentation strategy described in Section 5.3. All other settings for these experiments are identical to those of the class incremental learning experiments on ImageNet described in Appendix D.1.1.

D.1.5 Performance Impact of a Cold Start

The experiments in Section 5.6.4 utilize the same experimental setup for CIFAR100 as described in Section 5.6.1 and Appendix D.1.1. All results are recorded in terms of Top-1 Ω_{all} using the same offline normalization model from 5.6.1 that achieves a Top-1 accuracy of 78.61%. The only difference between the two experimental setups is that model accuracy is recorded after every new class is observed during the streaming process, whereas experiments in Section 5.6.1 only record model accuracy after every 20-class batch. Within Section 5.6.4, both the proposed methodology and all baselines are allowed to retain the full dataset within the replay buffer. The proposed methodology does not perform and quantization or resizing on the images. All hyperparameter settings are the same as Appendix D.1.1 for all methodologies.

D.1.6 Different Network Architectures

The experiments with different network architectures in Section 5.6.4 follow the same settings as the class-incremental streaming experiments on ImageNet provided in Section 5.6.4; see Appendix D.1.1 for further details. However, the ResNet18 architecture used in previous experiments is substituted for several different model architectures. The experimental results presented in Section 5.6.4 store the replay buffer on disk using JPEG compression, and a replay buffer capacity of 1.5Gb is adopted. All results are reported in terms of Top-5 Ω_{all}, and the offline models used for normalization (i.e., pre-trained models taken from the torchvision package [193]) achieve Top-5 accuracies listed in Table 23

D.1.7 Closing the Performance Gap with Offline Learning

Here, we present the details of experiments performed in Appendix D.2.6. Separate comparison are performed between incremental learning methodologies (i.e., iCarl and E2EIL) and streaming learning methodologies (i.e., REMIND) for class-incremental learning on the CIFAR100 dataset. In all cases, we use a ResNet18 model, but the architecture is slightly modified to yield best-possible performance on the CIFAR100 dataset, as demonstrated in [160]. Within these experiments, a class-specific ordering of the CIFAR100 dataset is induced that is used to ensure all experiments receive data in the same order. In all cases, the first 20 classes of CIFAR100 are used for base initialization.

For REMIND, we use the public implementation, but adapt it to utilize the modified ResNet18 architecture and perform training over CIFAR100. The learning rate is decayed from 0.1 to 0.001 for each class during streaming (i.e., we perform tests with different learning rates and fine that 0.1 still performs best), and we leverage random crops and manifold mixup within REMIND's feature representations. We allow all dataset examples to be stored within the replay buffer, due to the memory-efficiency of REMIND's memory indexing approach. Settings for PQ are kept identical to ImageNet experiments. We find that weight decay of 5×10^{-4} yields the best performance when evaluated using grid search over a validation set.

For iCarl and E2EIL, we adopt the official, public implementations of both approaches. Because the CIFAR100 dataset is small, we allow both iCarl and E2EIL store the full dataset within the replay buffer (i.e., we do not enforce a fixed buffer size) during the online learning process. To tune hyperparameters in the streaming setting, we perform a grid search over a validation set of CIFAR100 to obtain the optimal settings. We train iCarl and E2EIL in the typical, incremental fashion so that the impact of class imbalance can be observed. Within the incremental learning process, we use 20-class subsets of CIFAR100 as each sequential batch during the learning process. From here, we adopt identical hyperparameter settings as the original publication to replicate the original experimental settings.

D.2 Further Experimental Results

D.2.1 Data Augmentation Policies

We test all combinations of the Mixup, Cutmix, AutoAugment, and RandAugment data augmentation strategies to arrive at the final, optimal augmentation policy used within the proposed methodology for CSSL. All experiments perform random crops and flips in addition to other augmentation strategies. The

MU	CM	AA	RA	Top-1 Ω_{all}
				0.689 ± 0.001
✓				0.807 ± 0.012
	✓			0.863 ± 0.020
✓	✓			0.877 ± 0.004
		✓		0.822 ± 0.037
✓		✓		0.886 ± 0.018
	✓	✓		0.852 ± 0.019
✓	✓	✓		$\mathbf{0.886 \pm 0.002}$
✓			✓	0.856 ± 0.025
	✓		✓	0.857 ± 0.008
✓	✓		✓	0.875 ± 0.005

Table 24: Performance of the proposed methodology for CSSL with numerous augmentation policies that combine Mixup (MU), CutMix (CM), AutoAugment (AA), and RandAugment (RA) for class-incremental streaming experiments on CIFAR100. All tests use random clops and flips, and ✓ indicates the use of a particular augmentation technique.

comparison of different data augmentation policies is performed with class-incremental learning experiments on CIFAR100. These experiments adopt identical experimental settings as CIFAR100 experiments in Section 5.6.1; see Appendix D.1.1 for further details.

For these experiments, the entire dataset is allowed to be stored in the replay buffer, and the replay buffer is kept in main memory without the use of any compression schemes (i.e., no integer quantization or resizing). All reported results represent average performance recorded over three independent trials. For Mixup and Cutmix, we perform experiments using all combinations of α values in the set $\{0.1, 0.4, 0.8, 1.0, 1.2\}$. We leverage the CIFAR100 AutoAugment policy, and experiments are performed with RandAugment using 1-2 augmentations and all magnitudes in the set $\{4, 9, 14\}$. All combinations of considered data augmentation methodologies are tested, and results with best-performing hyperparameters are presented. These results inform the data augmentation policy that is adopted within the proposed methodology for CSSL used in later experiments.

The results of these experiments, reported in terms of Top-1 Ω_{all}, are provided in Table 24, where it can be seen that the best results are achieved by combining Mixup, Cutmix, and AutoAugment techniques into a single augmentation policy. Interestingly, this augmentation policy provides a nearly 20% improvement in absolute Ω_{all} compared to a simple crop and flip augmentation strategy, *thus demonstrating the massive impact of proper data augmentation on streaming performance*.

D.2.2 Memory Efficient Replay.

The proposed methodology for CSSL compresses replay examples using integer quantization (i.e., on the `uint8` pixels of each image) and image resizing. Within this section, we analyze the performance impact of this strategy in class-incremental streaming experiments on CIFAR100 using the same experimental

Quant.	Resize Ratio	Top-1 Ω_{all}
8 bit	100%	0.810 ± 0.009
6 bit	100%	0.822 ± 0.014
4 bit	100%	0.813 ± 0.025
8 bit	66%	0.819 ± 0.001
8 bit	50%	0.764 ± 0.029
6 bit	66%	$\mathbf{0.826 \pm 0.002}$
4 bit	66%	0.715 ± 0.038

Table 25: Performance of the proposed methodology on class-incremental streaming experiments with CIFAR100, where different levels of quantization and resizing are used when adding examples into the replay buffer.

Eviction Strategy	Top-1 Ω_{all}
Mixup [264]	0.746 ± 0.019
CutMix [259]	0.778 ± 0.014
SamplePairing [125]	0.735 ± 0.023

Table 26: Performance of CSSL when various interpolation strategies are used to combine replay examples instead of evicting them.

setup as outlined in Section 5.6.1. All combinations of quantizing `uint8` pixels to six or four bits[44] and resizing images to 66% or 50% of their original area are tested. For each possible augmentation strategy, the corresponding experiment stores the maximum number of images within the replay buffer without exceeding 30Mb of storage. Three separate trails are performed for each possible compression strategy, and average results across the three trials are reported. These results inform the optimal strategy of reducing memory overhead for the replay mechanism used within the proposed methodology for CSSL.

Adopting a fixed buffer capacity of 30Mb, we explore different levels of quantization and resizing within the replay buffer.[45] The results are reported in terms of Top-1 Ω_{all} in Table 25, where it can be seen that the best performance is achieved by quantizing pixel values to 6 bits and resizing images to 66% of their original area (i.e., the exact strategy adopted for CIFAR100 experiments in Section 5.6.1).

Interestingly, the proposed methodology is robust to high levels of quantization; e.g., using 4 bit quantization slightly exceeds the performance of storing full images for replay. Similar behavior is observed on ImageNet, where 4-bit quantization is found to outperform 6-bit quantization given a fixed buffer capacity. Moreover, resizing images beyond 66% of their original area noticeably degrades performance; e.g., 50% resizing without quantization yields noticeably degraded performance in Table 25. For datasets with higher-resolution images, we observe that performance is even more sensitive to the resizing ratio, leading us to adopt a ratio of 75% within ImageNet and Core50 experiments in Sections 5.6.1 and 5.6.2.

[44] The integer quantization procedure is simulated by integer dividing each pixel value by 2^{8-b}, where b is the number of bits present in the quantized pixel values, then re-scaling the result into the range $[0, 256)$.

[45] When images are compressed, more images are added into the replay buffer to reach the 30Mb capacity.

Buffer Capacity	Random	Reservoir
30Mb	0.810 ± 0.019	0.709 ± 0.099
60Mb	0.855 ± 0.011	0.764 ± 0.010
90Mb	0.889 ± 0.024	0.795 ± 0.004
120Mb	0.874 ± 0.009	0.830 ± 0.001

Table 27: Comparison of different strategies for maintaining the replay buffer for class-incremental CSSL experiments on CIFAR100.

D.2.3 Consolidating Replay Examples

In addition to the approaches for memory efficient replay described in Section 5.3, we attempt to use data interpolation methods to consolidate examples within the replay buffer. In particular, when a replay example is about to be evicted from the replay before, interpolation is performed between two replay examples, thus consolidating two entries in the replay buffer into one. Such an approach avoids evicting examples within the replay buffer without increasing memory overhead. In particular, we perform experiments with Mixup, CutMix, and SamplePairing interpolation strategies[46] for class-incremental learning on the CIFAR100 dataset, adopting the same experimental setting as described in Section 5.6.1. We use a fixed replay buffer capacity of 30Mb and report results of these experiments in terms of Top-1 Ω_{all} within Table 26. As can be seen, these eviction strategies are outperformed by the quantization and resizing strategies utilized in Section 5.6.4. We also consider setting a fixed number of times a certain example can be interpolated, such that a replay examples is evicted after it has been interpolated more than two or three times, but such an approach still does not meaningfully improve performance and introduces an extra hyperparameter.

D.2.4 Alternative Sampling Approaches

The methodology described in Section 5.3 assumes that, when a new example is added into an already-full replay buffer, an example must be removed via `ReplayEvict` in Algorithm 5. Within the experiments in Section 5.6, a random eviction procedure is adopted as described in Section 5.3. However, such random maintenance of the replay buffer is not the only option. Namely, different approaches such as herding selection [30, 119, 202] or reservoir sampling [5, 34] could be adopted instead to add and remove elements from the replay buffer in a more sophisticated manner.

Although herding selection is a popular method in incremental learning literature [30, 119, 202] for determining which examples within each incremental batch should be added into the replay buffer, it is not a viable approach given the restrictions of CSSL. Namely, herding selection requires that, for each class, a mean feature vector should be computed across all available data. Then, a representative subset of exemplars is selected within that class, such that the mean feature vector of that subset closely matches the overall mean feature vector. Such an approach is easily adoptable within incremental learning, as one can compute mean feature vectors and relevant exemplars for the disjoint classes that appear within each incremental batch.

[46]The α values used for Mixup and Cutmix are tuned by searching over the set $\{0.1, 0.4, 0.8, 1.0, 1.2\}$ using a hold-out validation set, and we present the results of the best-performing α for each method. By definition, Sample Pairing always takes an average between two images (i.e., there are no hyperparameters to tune).

`Full`	`Split`	`Sep`	`Sum`	Top-1 Accuracy MIT Scenes	 CUB-200	 Flowers	 FGVC	Average Top-1 Accuracy
✓		✓		**56.11**	60.79	**88.08**	45.22	**62.55**
	✓	✓		53.13	57.232	84.50	36.78	57.91
✓			✓	54.40	**62.12**	84.39	**49.09**	62.50
	✓		✓	54.48	60.32	86.57	43.05	61.11

Table 28: Multi-task streaming learning performance of CSSL with different replay strategies.

Further, herding selection can be modified to the streaming setting (i.e., based on a greedy approach) if fixed feature representations are available for each input image throughout the streaming process. However, within CSSL, the entire network is updated during streaming, meaning that the feature respresentations of data change over time, making the application of herding selection difficult.

Despite the difficulty of adopting herding selection within CSSL, we compare the performance of our random eviction policy with reservoir sampling with class-incremental learning experiments on the CIFAR100 dataset, where we adopt the same setup as described in Section 5.6.1. Reservoir sampling adds items into the replay buffer until the buffer is full. Assuming the capacity of the replay buffer is C examples, once the k-th example is received (i.e., $k > C$), we sample a random integer i in the set $\{0, 1, \dots, k-1\}$. If $i \in \{0, 1, \dots, k-1\}$, then the i-th example in the replay buffer is replaced with the k-th example (i.e., the new data). Otherwise, the k-th example is discarded. The performance of reservoir sampling in comparison to the random eviction strategy described in Section 5.3 is provided in Table 27, where we measure performance across multiple different buffer capacities (i.e., 150Mb is full capacity for CIFAR100). As can be seen, the more sophisticated reservoir sampling approach is outperformed by the random eviction strategy outlined in Section 5.3.

D.2.5 Different Strategies for Multi-Task Streaming Learning

Within Section 5.6.3, we generalize the replay strategy of CSSL and REMIND to the multi-task streaming learning domain. In this domain, each model has multiple classification layers and replay buffers (i.e., one for each dataset).[47] Thus, the previously-used strategy for replay—sampling a fixed number of replay samples to be included with each new data example during an online update—is no longer valid, as one must incorporate replay examples from previous tasks to avoid catastrophic forgetting.

The main strategies that were explored for multi-task replay were differentiated by $i)$ whether each task is updated with B replay examples or if B total replay examples are split between each task (i.e., `Full` or `Split`) and $ii)$ whether each task performs a separate update or if gradients are summed over all tasks to perform a single update (i.e., `Sep` or `Sum`). Considering these options forms four different strategies for multi-task replay. Using a fixed replay size of $B = 100$ and adopting the experimental settings explained in Appendix D.1.3 for multi-task streaming learning, these different strategies for multi-task streaming learning are explored for CSSL in Table 28. Models are initialized with ImageNet pre-trained parameters, but we do

[47]The separate replay buffers for each dataset could be considered as one replay buffer that encompasses all datasets and the resulting algorithm and discussion would be the same.

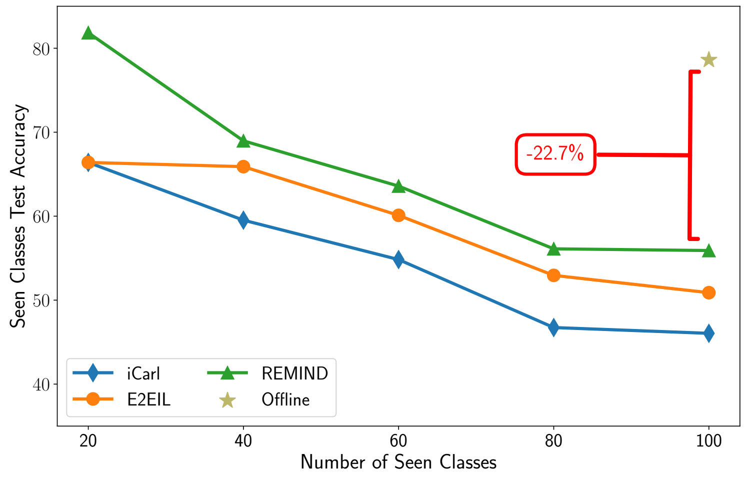

Figure 29: Top-1 test accuracy for ResNet18 trained on CIFAR100 with both online and offline approaches. Accuracy is measured over "seen" classes at different points during online learning.

not perform any fine-tuning on the MIT Indoor dataset prior to streaming.

As can be seen in Table D.2.5, the best performing strategy utilizes the full replay size for each task during the online update and performs a separate model update for each task. A strategy of using the full replay size for each task but performing a single update that sums the gradients of each task together performs similarly. However, this strategy is slightly worse than performing a separate update for each task. Thus, within Section 5.6.3, we adopt the strategy of, at each online update, sampling B replay examples—or $B - 1$ replay examples for the current task, plus the new example—for each of the tasks that have been seen so far and performing a separate model update for each task. The added computation of performing a separate update for each task is minimal relative to performing a single update across all tasks.

D.2.6 Closing the Performance Gap with Offline Learning

We claim that streaming learning methodologies have the potential for drastically-improved performance, but cannot be derived via simple modifications to existing techniques. To substantiate this claim, we perform class-incremental learning experiments on CIFAR100 [144] with two batch-incremental methods—iCarl [202] and End-to-End Incremental Learning (E2EIL) [30]—and a state-of-the-art streaming approach—REMIND [103]; see Appendix D.1.7 for details. Performance is measured using Ω_{all} [102] and displayed in Figure 29.

As can be seen, a performance gap exists between online and offline-trained models; e.g., in Figure 29, the final Top-1 accuracy of REMIND is $> 20\%$ below that of offline training. To be of use to most practitioners, this performance gap between online and offline training must be reduced—*no viable, real-time alternatives to offline training yet exist.* Considering possible sources of such a performance gap, one may look to the freezing of network layers within existing approaches to online learning. Such an approach is shown to significantly degrade network performance in Section 5.4. Thus, an approach with the potential to perform more comparably to offline training likely should not freeze large portions of network parameters.

Interestingly, many incremental learning approaches exist that perform end-to-end training [30, 119, 245]. Thus, one may wonder whether a high-quality streaming approach—one that comes closer to matching

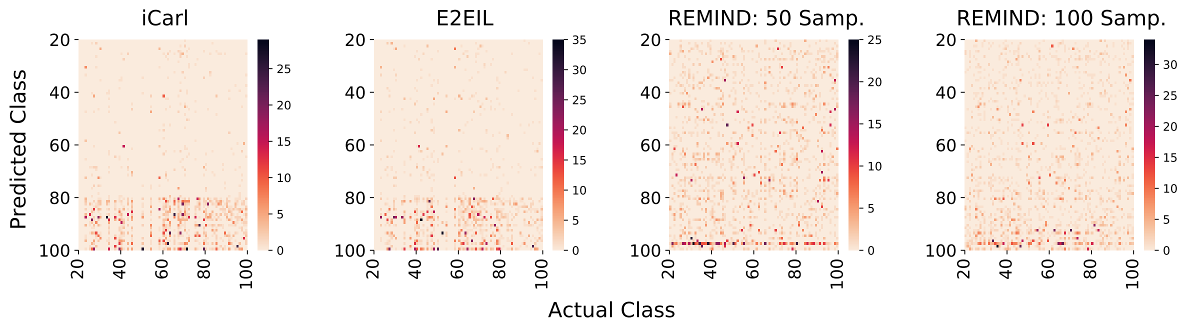

Figure 30: Confusion matrices (excluding base initialization data) for incremental and streaming learning models after class-incremental CIFAR100 training. Incremental learning techniques (iCarl and E2EIL) are biased towards recently-observed classes (see the many incorrect predictions for classes 80-100 in the two leftmost plots). In contrast, incorrect predictions for REMIND are concentrated on the last class observed during streaming (see the line of incorrect predictions at the bottom of third plot). This bias starts getting corrected when 100 replay samples—instead of 50—are used during each online update (see the rightmost plot).

Method	PQ Encoding Time	Forward Pass Time	Backward Pass Time	Parameter Update Time
CSSL	-	3.67 ± 0.45	5.41 ± 0.97	1.53 ± 0.06
REMIND	23.37 ± 1.13	2.65 ± 0.02	1.62 ± 0.03	0.52 ± 0.01
REMIND + Extra Params	23.37 ± 1.13	2.95 ± 0.12	1.84 ± 0.09	0.69 ± 0.04

Table 29: Timing metrics (in milliseconds) for performing model inference and updates with different streaming methodologies on ImageNet.

offline learning performance—could be developed by simple modifications to such techniques. Unfortunately, these techniques, which employ replay, bias correction, and knowledge distillation to prevent catastrophic forgetting, are known to perform poorly in the streaming domain [103]. To understand why this is the case, it should be noted that knowledge distillation is known to provide minimal added benefit when combined with replay mechanisms [20, 21]. Furthermore, we find that correcting bias in the network's classification layer is unnecessary for streaming learning. Though batch-incremental techniques are biased towards recently-observed classes, REMIND is only biased towards a single class (i.e., the last streamed class), which is easily corrected by increasing the number of replay samples observed during each online update; see Figure 30 and Appendix D.1.7 for experimental details.[48]

Such observations reveal that findings relevant to batch-incremental learning may not translate to the streaming setting. Additionally, a better streaming methodology cannot be derived by simply adopting and modifying established techniques for batch incremental learning. To close the performance gap with offline learning, streaming techniques likely need to avoid freezing network parameters and leverage more effective forms of replay, as other techniques like knowledge distillation and bias correction provide minimal benefit.

[48]Many incremental learning works have been proposed to correct such bias towards recently-observed classes [119, 245]. We emphasize that our focus here is not to compare streaming learning to the most recent approaches for incremental learning, but rather to demonstrate that bias towards recent classes is not a consideration for the streaming domain that we consider.

D.2.7 Computational Impact of Full Plasticity

Given that CSSL makes all model parameters trainable, one may wonder about the impact of such an approach on the computational complexity of performing updates and inferences with the underlying model throughout streaming. To better understand the complexity impact of full plasticity, we measure the time taken to $i)$ perform a forward pass, $ii)$ perform a backward pass, and $iii)$ update model parameters with CSSL over several streaming iterations. For comparison, we perform the same test with REMIND, which freezes a majority of the underlying network's early layers.

These timing tests use a ResNet18 model and are performed on the ImageNet dataset using the same procedure described in Appendix D.1.1. At each iteration, 100 replay examples are sampled for both CSSL and REMIND. For REMIND, we also perform supplemental test in which the underlying model has added layers to make the number of trainable parameters equal to that of CSSL. In all cases, we measure timing across five streaming iterations and report the average time; see Table 29.

CSSL has a more expensive forward pass, backward pass, and parameter update in comparison to REMIND, due to the fact that all network layers are updated by CSSL. However, REMIND encodes the output of the network's frozen feature extractor with a product quantization (PQ) [130] module, which is quite slow in comparison to the other operations. As such, when data is passed through the REMIND model—aside from replay examples for which the model's encoded hidden representations can be cached—it must be passed through the frozen portion of the network, encoded via PQ, then passed through the remaining, trainable layers, making the total inference time of REMIND an order of magnitude slower than that of CSSL.

D.3 Adapting Baselines to CSSL.

CSSL is capable of randomly initializing the full network—and all network modules—immediately prior to the commencement of streaming; see Section 5.3. The baseline methodologies used within Section 5.6.1 (i.e., ExStream, Deep SLDA, and REMIND), however, all utilize pre-trained networks within their respective streaming algorithms. In particular, ExStream [102] and Deep SLDA [104] train only the final, fully-connected layer of the neural network, while REMIND trains the last few convolutional layers and the final fully-connected layer during streaming—all other network layers are fixed either by using pre-trained network parameters or performing offline training over a subset of data during base initialization.

To be capable of starting streaming from a random initialization, baseline methodologies would either have to $i)$ allow nearly all (or a majority of) network layers to remain random throughout streaming or $ii)$ be adapted to train the network end-to-end. Forcing network layers to remain random throughout streaming catastrophically degrades performance, and adapting such methodologies to perform end-to-end training is highly non-trivial, as a majority of techniques within each methodology would break down (e.g., compressing the feature representations with PQ, clustering feature vectors, computing feature covariance, etc.). As such, we simply compare CSSL to baseline approaches in their original form, without forcing baselines to conform a cold-start setting (i.e., baselines do not satisfy rule four from Section 5.3).

D.4 Technical Results

Within this section, we outline the technical theoretical results (i.e., Lemmas and Corollaries) that are established in the process of proving Theorem 6. Again, full proofs of each result are deferred to Section D.5.

Lemma 10. *Assume Assumption 3 holds and $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$. Consider iterates $\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ obtained via Algorithm 5 with B replay examples taken from buffer $\mathcal{R}$ per iteration with $\eta = \mathcal{O}(m^{-\frac{3}{2}})$. Given sufficient overparamterization characterized by*

$$m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$$

We have

$$\sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^{(i)}) \leq \sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^{\star}) + \mathcal{O}\left(LR^2 m^{-\frac{1}{2}}\right) + \mathcal{O}\left(nm^{-\frac{1}{2}}BL(1+\Xi)\right)$$

for $\mathbf{W}^{\star} \in \mathcal{B}(\mathbf{W}^{(0)}, R/m)$ with $R > 0$ and probability at least $1 - \mathcal{O}(nBL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

Lemma 11. *Given Assumption 3 and taking $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$, we have that*

$$\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$$

over n successive iterations of Algorithm 5 with B replay samples taken from buffer $\mathcal{R}$ per iteration, arbitrary perturbation vectors applied to both new and replayed data, $\eta = \mathcal{O}(m^{-\frac{3}{2}})$, and sufficient overparameterization given by

$$m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$$

with probability at least $1 - \mathcal{O}(nLB)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

Lemma 12. *Assume Assumption 3 holds and $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$. Then, when $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, we have*

$$\begin{aligned} &\left|f_{\mathbf{W}'}(x_t+\xi) - f_{\mathbf{W}}(x_t+\xi) - \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t+\xi), \mathbf{W}' - \mathbf{W}\right\rangle\right| \\ &\quad \leq \mathcal{O}\left(\sqrt{m\left(1+\|\xi\|_2^2\right)}\omega^{\frac{4}{3}}L^3\log(m)\right) \end{aligned}$$

for arbitrary perturbation vector ξ[49] and all $t \in [n]$ with probability at least $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

[49]Here, we do not assign a subscript ξ_t to the perturbation vector intentionally. This is because the perturbation vector used for this result is arbitrary. Namely, different perturbation vectors can be used for any $t \in [n]$ and both when the data is encountered newly or sampled for replay. The result holds for any perturbation vector chosen for each of these scenarios, where the final bound then depends on the factor $\|\xi\|_2^2$.

Corollary 1. *Assume Assumption 3 holds and* $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$*, then when* $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$*, we can extend Lemma 12 to show*

$$L_{(t,\xi)}(\mathbf{W}') - L_{(t,\xi)}(\mathbf{W}) \geq \sum_{l=1}^{L} \langle \nabla_{W_l} L_{(t,\xi)}(\mathbf{W}), W_l' - W_l \rangle - \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\omega^{\frac{4}{3}} L^3 \log(m)\right)$$

for all $t \in [n]$ *and arbitrary perturbation vector* ξ *with probability at least* $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

Lemma 13. *Assume Assumption 3 holds and* $\omega \leq \mathcal{O}\left(L^{-\frac{9}{2}}\log^{-3}(m)\right)$*. Then, for* $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ *we have*

$$\|g_{(t,l,\xi)} - g_{(t,l,\xi)}^{(0)}\|_2,\ \|h_{(t,l,\xi)} - h_{(t,l,\xi)}^{(0)}\|_2 \leq \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1+\|\xi\|_2^2)\log(m)}\right)$$

for all $t \in [n]$*,* $l \in [L-1]$*, and arbitrary perturbation vector* ξ *with probability at least* $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

Lemma 14. *Assume Assumption 3 holds and* $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$*. Then, with probability at least* $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ *we have for all* l *such that* $l \in [L-1]$ *and* $t \in [n]$

$$\left\| W_L' \prod_{r=l}^{L-1} \left(D_{(t,r,\xi)}' + D_{(t,r)}''\right) W_r' - W_L \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \right\|_2 \leq \mathcal{O}\left(\omega^{\frac{1}{3}} L^2 \sqrt{\log(m)}\right)$$

where $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$*,* $D_{i,r}'' \in [-1,1]^{m \times m}$ *is any random diagonal matrix with at most* $\mathcal{O}\left(m\omega^{\frac{2}{3}}L\right)$ *non-zero entries, and* ξ *is an arbitrary perturbation vector.*

Lemma 15. *Given Assumption 3, randomly initialized weights* $\mathbf{W}^{(0)}$*, and arbitrary perturbation vector* ξ *we have*

$$\left\|h_{(t,j,\xi)}^{(0)}\right\|_2^2 \in \left[1 - \|\xi\|_2^2, 1 + \|\xi\|_2^2\right]$$

with probability at least $1 - \mathcal{O}\left(nL\right) e^{-\Omega(m/L)}$ *for all* $t \in [n]$ *and* $j \in [L-1]$.

Corollary 2. *Given Assumption 3 and assuming* $\omega \leq \mathcal{O}(L^{-\frac{9}{2}}\log^{-3}(m))$*, we have for* $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ *and arbitrary perturbation vector* ξ

$$\|h_{(t,j,\xi)}\|_2 \leq \mathcal{O}\left(\sqrt{1+\|\xi\|_2^2}\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)} + 1\right)\right)$$

for all $t \in [n]$ *and* $j \in [L-1]$ *with probability at least* $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

Lemma 16. *Given Assumption 3 and $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$, the following can be shown when $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ for arbitrary perturbation vector ξ*

$$\|\nabla_{W_l} f_{\mathbf{W}}(x_t+\xi)\|_F\,, \left\|\nabla_{W_l} L_{(t,\xi)}(\mathbf{W})\right\|_F \leq \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\right)$$

for all $t \in [n]$ and $l \in [L-1]$ with probability at least $1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$.

D.5 Proofs

Within this section, we provide full proofs for all theoretical results derived within this work. We begin with an overview of further notation used within the analysis.

Notation. For some arbitrary set of weight matrices $\mathbf{W} = \{W_1, \dots, W_L\}$, we define $h_{(t,0,\xi)} = x_t + \xi$ and $h_{(t,l,\xi)} = \sigma(W_l h_{(t,l-1,\xi)})$ for $l \in [L-1]$ and arbitrary perturbation vector ξ. Similarly, we define $g_{(t,0,\xi)} = x_t + \xi$ and $g_{(t,l,\xi)} = W_l h_{(t,l-1,\xi)}$ for $l \in [L-1]$. For the weight matrices at initialization $\mathbf{W}^{(0)} = \{W_1^{(0)}, \dots W_L^{(0)}\}$, we would similarly define such vectors with the notation $h^{(0)}_{(t,l,\xi)}$ and $g^{(0)}_{(t,l,\xi)}$ (or $h'_{(t,l,\xi)}$, $g'_{(t,l,\xi)}$ for weight matrices $\mathbf{W}'$ and so on). We also define $D_{(t,l,\xi)} = \texttt{diag}(\mathbb{1}\{(W_l h_{(t,l,\xi)})_1 > 0\}, \dots, \mathbb{1}\{(W_l h_{(t,l,\xi)})_m > 0\})$ for all $t \in [n]$ and $l \in [L-1]$.

D.5.1 Proof of Theorem 6

Proof. For $t \in [n]$ and associated perturbation vectors ξ_t for $t \in [n]$, let $L^{0-1}_{(t,\xi_t)}(\mathbf{W}^{(t)}) = \mathbb{1}\left\{y_t \cdot f_{\mathbf{W}^{(t)}}(x_t+\xi_t) < 0\right\}$, where $L^{0-1}_{(t,\xi_t)}(\mathbf{W}^{(t)}) \leq 4L_{(t,\xi_t)}(\mathbf{W}^{(t)})$ due to properties of the cross entropy loss function (i.e., $\mathbb{1}\{z \leq 0\} \leq 4\ell(z)$). With this in mind, when $m > \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$ and $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$, we can invoke Lemma 10 to yield

$$\frac{1}{n}\sum_{t=1}^{n} L^{0-1}_{(t,\xi_t)}(\mathbf{W}^{(t)}) \leq \frac{4}{n}\sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^{\star}) + \mathcal{O}\left(\frac{LR^2}{nm^{\frac{1}{2}}}\right) + \mathcal{O}\left(m^{-\frac{1}{2}}BL(1+\Xi)\right)$$

with probability at least $1-\mathcal{O}(nBL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. Then, because the model is trained in a streaming fashion as described in Section 5.5.1, the t-th new data example is seen for the first time only at iteration t within the data stream. Thus, we have that weights $\mathbf{W}^{(t)}$ only depend on data examples $(x_1, y_1), \dots, (x_{t-1}, y_{t-1})$. In other words, $\mathbf{W}^{(t)}$ is independent of the t-th new data example (x_t, y_t). Thus, we can invoke an online-to-batch conversion argument (see Proposition 1 in [31]) to yield

$$\frac{1}{n}\sum_{t=1}^{n} L^{0-1}_{\mathcal{D}}(\mathbf{W}^{(t)}) \leq \frac{1}{n}\sum_{t=1}^{n} L^{0-1}_{(t,\xi_t)}(\mathbf{W}^{(t)}) + \sqrt{\frac{2\log(1/\delta)}{n}}$$

with probability at least $1-\delta$ for $\delta \in (0,1]$. From here, we define $\hat{\mathbf{W}}$ as a random entry chosen uniformly from the set $\{\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)}\}$. Thus, we have the following by definition

$$\frac{1}{n}\sum_{t=1}^{n} L_{\mathcal{D}}^{0-1}(\mathbf{W}^{(t)}) = \mathbb{E}\left[L_{\mathcal{D}}^{0-1}(\hat{\mathbf{W}})\right]$$

Then, the following can be derived by combining the above expressions

$$\begin{aligned}\mathbb{E}\left[L_{\mathcal{D}}^{0-1}(\hat{\mathbf{W}})\right] &\leq \frac{4}{n}\sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^{\star}) + \mathcal{O}\left(\frac{LR^2}{nm^{\frac{1}{2}}}\right) \\ &\quad + \mathcal{O}\left(m^{-\frac{1}{2}}BL(1+\Xi)\right) + \sqrt{\frac{2\log(\frac{1}{\delta})}{n}}\end{aligned} \tag{47}$$

with probability at least $1-\delta-\mathcal{O}(nBL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ for all $\mathbf{W}^{\star} \in \mathcal{B}(\mathbf{W}^{(0)}, {}^{R}\!/\!_{m})$. Now, we can compare the neural network function $f_{\mathbf{W}^{\star}}$ with $F_{\mathbf{W}^{(0)},\mathbf{W}^{\star}}$ as follows

$$L_{(t,\xi_t)}(\mathbf{W}^{\star}) \overset{(i)}{\leq} \ell(y_t \cdot F_{\mathbf{W}^{(0)},\mathbf{W}^{\star}}(x_t+\xi_t)) + \mathcal{O}\left(\sqrt{m\,(1+\Xi)}\omega^{\frac{4}{3}}L^3\log(m)\right)$$

where (i) holds from the 1-Lipschitz continuity of $\ell(\cdot)$ and Lemma 12 with probability at least $1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. Then, we can plug this inequality into (47) to yield

$$\begin{aligned}\mathbb{E}\left[L_{\mathcal{D}}^{0-1}(\hat{\mathbf{W}})\right] &\leq \frac{4}{n}\sum_{t=1}^{n} \ell[y_t \cdot F_{\mathbf{W}^{(0)},\mathbf{W}^{\star}}(x_t+\xi_t)] + \mathcal{O}\left(\sqrt{m\,(1+\Xi)}\omega^{\frac{4}{3}}L^3\log(m)\right) \\ &\quad + \mathcal{O}\left(\frac{LR^2}{nm^{\frac{1}{2}}}\right) + \mathcal{O}\left(m^{-\frac{1}{2}}BL(1+\Xi)\right) + \sqrt{\frac{2\log(\frac{1}{\delta})}{n}}\end{aligned}$$

Then, by taking an infimum over all $\mathbf{W}^{\star} \in \mathcal{B}(\mathbf{W}^{(0)}, {}^{R}\!/\!_{m})$, we get the following

$$\begin{aligned}\mathbb{E}\left[L_{\mathcal{D}}^{0-1}\left(\hat{\mathbf{W}}\right)\right] &\leq \inf_{f\in\mathcal{F}(\mathbf{W}^{(0)},{}^{R}\!/\!_{m})}\left(\frac{4}{n}\sum_{t=1}^{n}\ell(y_t\cdot f(x_t+\xi_t))\right) \\ &\quad + \mathcal{O}\left(\sqrt{m\,(1+\Xi)}\omega^{\frac{4}{3}}L^3\log(m)\right) + \mathcal{O}\left(\frac{LR^2}{nm^{\frac{1}{2}}}\right) \\ &\quad + \mathcal{O}\left(m^{-\frac{1}{2}}BL(1+\Xi)\right) + \sqrt{\frac{2\log({}^{1}\!/\!_{\delta})}{n}}\end{aligned}$$

From here, by noting that $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$ and $m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$, the expression can be simplified as follows

$$\begin{aligned}\mathbb{E}\left[L_{\mathcal{D}}^{0-1}\left(\hat{\mathbf{W}}\right)\right] \leq \inf_{f\in\mathcal{F}(\mathbf{W}^{(0)},R/m)}&\left(\frac{4}{n}\sum_{t=1}^{n}\ell(y_t\cdot f(x_t+\xi_t))\right)+\sqrt{\frac{2\log(1/\delta)}{n}}\\ &+\mathcal{O}\left(\frac{(1+\Xi)^{\frac{3}{4}}\sqrt{nB}}{L^2\log^{\frac{3}{2}}(m)}+\frac{R^2}{n^{\frac{3}{2}}B^{\frac{1}{2}}(1+\Xi)^{\frac{1}{4}}L^2\log^{\frac{3}{2}}(m)}\right.\\ &\left.+\frac{B^{\frac{1}{2}}(1+\Xi)^{\frac{3}{4}}}{n^{\frac{1}{2}}L^2\log^{\frac{3}{2}}(m)}\right)\end{aligned} \tag{48}$$

We then analyze the asymptotic portion of the expression above—highlighted in red—as follows.

$$\begin{aligned}&\mathcal{O}\left(\frac{(1+\Xi)^{\frac{3}{4}}\sqrt{nB}}{L^2\log^{\frac{3}{2}}(m)}+\frac{R^2}{n^{\frac{3}{2}}B^{\frac{1}{2}}(1+\Xi)^{\frac{1}{4}}L^2\log^{\frac{3}{2}}(m)}+\frac{B^{\frac{1}{2}}(1+\Xi)^{\frac{3}{4}}}{n^{\frac{1}{2}}L^2\log^{\frac{3}{2}}(m)}\right)\\ &\quad=\mathcal{O}\left(\frac{(1+\Xi)B(n^2+1)+R^2}{L^2\log^{\frac{3}{2}}(m)nB^{\frac{1}{2}}(1+\Xi)^{\frac{1}{4}}}\right)\\ &\quad=\mathcal{O}\left(\frac{(1+\Xi)^{\frac{3}{4}}B^{\frac{1}{2}}n}{L^2\log^{\frac{3}{2}}(m)}\right)+\mathcal{O}\left(\frac{R^2}{L^2\log^{\frac{3}{2}}(m)nB^{\frac{1}{2}}(1+\Xi)^{\frac{1}{4}}}\right)\\ &\quad\overset{(i)}{\leq}\mathcal{O}\left(\frac{(1+\Xi)^{\frac{3}{4}}B^{\frac{1}{2}}n+R^2}{L^2\log^{\frac{3}{2}}(m)}\right)\end{aligned}$$

where (i) holds due to the fact that $nB^{\frac{1}{2}}(1+\Xi)^{\frac{1}{4}} > 1$. Then, by substituting this simplified asymptotic expression, we have

$$\mathbb{E}\left[L_{\mathcal{D}}^{0-1}\left(\hat{\mathbf{W}}\right)\right] \leq \inf_{f\in\mathcal{F}(\mathbf{W}^{(0)},R/m)}\left(\frac{4}{n}\sum_{t=1}^{n}\ell(y_t\cdot f(x_t+\xi_t)\right)+\sqrt{\frac{2\log(1/\delta)}{n}}+\mathcal{O}\left(\frac{(1+\Xi)^{\frac{3}{4}}B^{\frac{1}{2}}n+R^2}{L^2\log^{\frac{3}{2}}(m)}\right)$$

where the asymptotic portion of the expression can be made arbitrarily small by increasing the value of m. Realizing that this result holds with probability $1-\delta-\mathcal{O}(nBL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)} \approx 1-\delta$ as the value of m increases yields the desired result. □

D.5.2 Proof of Lemma 10

Proof. We assume $\mathbf{W}^{(0)}$ is initialized as described in Section 5.5.1, then updated according to Algorithm 5 with B replay samples taken from buffer $\mathcal{R}$ at each iteration and distinct, arbitrary data perturbation vectors—representing data augmentation—applied to both new and replayed data throughout streaming. For $R > 0$, we have $\mathbf{W}^\star \in \mathcal{B}(\mathbf{W}^{(0)}, R/m)$, where $\mathbf{W}^\star \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ whenever $m \geq \mathcal{O}(L^6\log^3(m))$ (i.e., R is a constant that does not appear in the asymptotic bound), which is looser than the overparameterization requirement within Lemma 11. Similarly, by Lemma 11, we have $\mathbf{W}^{(0)}, \ldots, \mathbf{W}^{(n)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ with probability at least $1-\mathcal{O}(nLB)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ whenever $m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$ and $\eta = \mathcal{O}(m^{-\frac{3}{2}})$.

Thus, for $\mathbf{W}^{(t)}, \mathbf{W}^\star \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, by Corollary 1 we have with probability at least $1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ that

$$\begin{aligned}
L_{(t,\xi_t)}(\mathbf{W}^{(t)}) - L_{(t,\xi_t)}(\mathbf{W}^\star) &\leq \left\langle \nabla_{\mathbf{W}^{(t)}} L_{(t,\xi_t)}(\mathbf{W}^{(t)}), \mathbf{W}^{(t)} - \mathbf{W}^\star \right\rangle \\
&\quad + \mathcal{O}\left(\sqrt{m(1+\Xi)}\omega^{\frac{4}{3}} L^3 \log(m)\right) \\
&= \sum_{l=1}^{L} \left\langle \nabla_{W_l^{(t)}} L_{(t,\xi_t)}(\mathbf{W}^{(t)}), W_l^{(t)} - W_l^\star \right\rangle \\
&\quad + \mathcal{O}\left(\sqrt{m(1+\Xi)}\omega^{\frac{4}{3}} L^3 \log(m)\right)
\end{aligned} \tag{49}$$

We can focus on bounding the red term within the expression above as follows

$$\begin{aligned}
&\sum_{l=1}^{L} \left\langle \nabla_{W_l^{(t)}} L_{(t,\xi_t)}(\mathbf{W}^{(t)}), W_l^{(t)} - W_l^\star \right\rangle \\
&\overset{i}{=} \sum_{l=1}^{L} \frac{1}{\eta} \left\langle W_l^{(t)} - W_l^{(t+1)} - \eta \sum_{(x_{\text{rep}}, y_{\text{rep}})\sim\mathcal{S}_t} \nabla_{W_l^{(t)}} L_{(x_{\text{rep}}, y_{\text{rep}}, \xi_{\text{rep}})}(\mathbf{W}^{(t)}), W_l^{(t)} - W_l^\star \right\rangle \\
&\overset{ii}{\leq} \sum_{l=1}^{L} \frac{1}{2\eta} \left(\left\| W_l^{(t)} - W_l^{(t+1)} - \eta \sum_{(x_{\text{rep}}, y_{\text{rep}})\sim\mathcal{S}_t} \nabla_{W_l^{(t)}} L_{(x_{\text{rep}}, y_{\text{rep}}, \xi_{\text{rep}})}(\mathbf{W}^{(t)}) \right\|_F^2 + \left\| W_l^{(t)} - W_l^\star \right\|_F^2 \right. \\
&\qquad \left. - \left\| W_l^{(t+1)} - W_l^\star + \eta \sum_{(x_{\text{rep}}, y_{\text{rep}})\sim\mathcal{S}_t} \nabla_{W_l^{(t)}} L_{(x_{\text{rep}}, y_{\text{rep}}, \xi_{\text{rep}})}(\mathbf{W}^{(t)}) \right\|_F^2 \right) \\
&= \sum_{l=1}^{L} \frac{1}{2\eta} \left(\left\| \eta \nabla_{W_l^{(t)}} L_{(t,\xi_t)}(\mathbf{W}^{(t)}) \right\|_F^2 + \left\| W_l^{(t)} - W_l^\star \right\|_F^2 \right. \\
&\qquad \left. - \left\| W_l^{(t+1)} - W_l^\star + \eta \sum_{(x_{\text{rep}}, y_{\text{rep}})\sim\mathcal{S}_t} \nabla_{W_l^{(t)}} L_{(x_{\text{rep}}, y_{\text{rep}}, \xi_{\text{rep}})}(\mathbf{W}^{(t)}) \right\|_F^2 \right) \\
&\overset{iii}{\leq} \sum_{l=1}^{L} \frac{1}{2\eta} \left(\left\| \eta \nabla_{W_l^{(t)}} L_{(t,\xi_t)}(\mathbf{W}^{(t)}) \right\|_F^2 + \left\| W_l^{(t)} - W_l^\star \right\|_F^2 \right. \\
&\qquad \left. - \left\| W_l^{(t+1)} - W_l^\star \right\|_F^2 + \left\| \eta \sum_{(x_{\text{rep}}, y_{\text{rep}})\sim\mathcal{S}_t} \nabla_{W_l^{(t)}} L_{(x_{\text{rep}}, y_{\text{rep}}, \xi_{\text{rep}})}(\mathbf{W}^{(t)}) \right\|_F^2 \right) \\
&\overset{iv}{\leq} \sum_{l=1}^{L} \frac{1}{2\eta} \left(\left\| W_l^{(t)} - W_l^\star \right\|_F^2 - \left\| W_l^{(t+1)} - W_l^\star \right\|_F^2 \right) \\
&\qquad + \mathcal{O}\left(\eta B L m (1+\Xi)\right)
\end{aligned}$$

where i follows from (21), ii holds from the identity that $2\langle A, B\rangle \leq \|A\|_F^2 + \|B\|_F^2 - \|A-B\|_F^2$, and iii holds due to the lower triangle inequality, and iv holds due to the upper triangle inequality and Lemma 16 with probability at least $1-\mathcal{O}(nBL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ by taking union bound across all data, replay examples, and network layers. Plugging this bound into (49), we get

$$L_{(t,\xi_t)}(\mathbf{W}^{(t)}) - L_{(t,\xi_t)}(\mathbf{W}^\star) \leq \sum_{l=1}^{L} \frac{1}{2\eta}\left(\left\|W_l^{(t)} - W_l^\star\right\|_F^2 - \left\|W_l^{(t+1)} - W_l^\star\right\|_F^2\right)$$
$$+ \mathcal{O}\left(\eta BLm(1+\Xi) + \sqrt{m(1+\Xi)}\omega^{\frac{4}{3}}L^3\log(m)\right)$$

Then, telescoping this expression over $t \in [n]$ yields

$$\begin{aligned}
\sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^{(t)}) &\leq \sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^\star) + \sum_{l=1}^{L}\frac{\left\|W_l^{(0)} - W_l^\star\right\|_F^2}{2\eta} \\
&\quad + \mathcal{O}\left(n\eta BLm(1+\Xi) + n\sqrt{m(1+\Xi)}\omega^{\frac{4}{3}}L^3\log(m)\right) \\
&\overset{i}{\leq} \sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^\star) + \frac{LR^2}{2\eta m^2} \\
&\quad + \mathcal{O}\left(n\eta BLm(1+\Xi) + n\sqrt{m(1+\Xi)}\omega^{\frac{4}{3}}L^3\log(m)\right) \\
&\overset{ii}{\leq} \sum_{t=1}^{n} L_{(t,\xi_t)}(\mathbf{W}^\star) + \mathcal{O}\left(LR^2m^{-\frac{1}{2}}\right) + \mathcal{O}\left(nm^{-\frac{1}{2}}BL(1+\Xi)\right)
\end{aligned}$$

where i follows from the fact that $\mathbf{W}^\star \in \mathcal{B}(\mathbf{W}^{(0)}, {}^{R}\!/_{m})$ and ii holds for sufficiently small ω with $\eta = \frac{\kappa}{m^{\frac{3}{2}}}$, where κ is a small, positive constant. Thus, the desired result holds with probability at least $1 - \mathcal{O}(nBL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ with overparameterization given by the expression below.

$$m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6\log^3(m)\right)$$

□

D.5.3 Proof of Lemma 11

Proof. We assume $\mathbf{W}^{(0)}$ is initialized as described in Section 5.5.1 and updated according to Algorithm 5 with B replay examples sampled uniformly from the replay buffer $\mathcal{R}$ at each update and arbitrary perturbation vectors—representing data augmentation—applied separately to both new and replayed data throughout streaming. We take $\omega = C_1L^{-6}\log^{-3}(m)$ such that C_1 is a small enough constant to satisfy assumptions on ω in Lemma 16. From here, we can show that $\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ through induction.

Base Case. The base case $\mathbf{W}^{(0)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ holds trivially.

Inductive Case. Assume that $\mathbf{W}^{(t)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$. By Lemma 16, we have

$$\left\|\nabla_{W_l}L_{(t,\xi_t)}(\mathbf{W})\right\|_F \leq \mathcal{O}\left(\sqrt{m(1+\|\xi_t\|_2^2)}\right) \tag{50}$$

with probability at least $1 - e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. Denoting the indices of data elements within our replay buffer as

$\mathcal{R}$, we can then show the following over iterations of Algorithm 5.

$$\begin{aligned}
&\left\|W_l^{(t+1)} - W_l^{(0)}\right\|_F \\
&\overset{i}{\leq} \sum_{j=1}^{t} \left\|W_l^{(j+1)} - W_l^{(j)}\right\|_F \\
&\overset{ii}{=} \sum_{j=1}^{t} \left\|-\eta \left(\sum_{(x_{\text{rep}},y_{\text{rep}})\sim\mathcal{S}_t} \nabla_{W_l} L_{(x_{\text{rep}},y_{\text{rep}},\xi_{\text{rep}})}\left(\mathbf{W}^{(j)}\right) + \nabla_{W_l} L_{(j,\xi_j)}\left(\mathbf{W}^{(j)}\right)\right)\right\|_F \\
&\overset{iii}{\leq} \eta \sum_{j=1}^{t} \left(\sum (x_{\text{rep}}, y_{\text{rep}}) \sim \mathcal{S}_t \left\|\nabla_{W_l} L_{(x_{\text{rep}},y_{\text{rep}},\xi_{\text{rep}})}\left(\mathbf{W}^{(j)}\right)\right\| + \left\|\nabla_{W_l} L_{(j,\xi_j)}\left(\mathbf{W}^{(j)}\right)\right\|\right) \\
&\overset{iv}{\leq} \mathcal{O}\left(\eta n B \left(\sqrt{m\,(1+\Xi)}\right)\right)
\end{aligned}$$

where i and iii hold due to the upper triangle inequality, ii holds from (21), and iv holds due to (50) and the definition of Ξ. Thus, for $\omega = C_1 L^{-6} \log^{-3}(m)$, if we set $\eta = \frac{\kappa}{m^{\frac{3}{2}}}$ we have

$$\begin{aligned}
\left\|W_l^{(t+1)} - W_l^{(0)}\right\|_F &\leq \mathcal{O}\left(\eta n B \sqrt{m\,(1+\Xi)}\right) \\
&= \frac{\kappa n B \sqrt{1+\Xi}}{\sqrt{m}} \overset{i}{\leq} \omega
\end{aligned}$$

for some small enough absolute constant κ and $m \geq \mathcal{O}\left(nB\sqrt{1+\Xi}L^6 \log^3(m)\right)$. Thus, the inductive step is complete and we have $\mathbf{W}^{(0)}, \dots, \mathbf{W}^{(n)} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ with probability at least $1 - \mathcal{O}(nLB)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ by taking a union bound over all data examples, replay examples, and network layers. □

D.5.4 Proof of Lemma 12

Proof. We consider some fixed $t \in [n]$ with perturbation vector ξ and two weight matrices $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, where $\mathbf{W}^{(0)}$ is initialized as described in Section 5.3.1. It should be noted that ξ is an arbitrary perturbation vector that can be different for any $t \in [n]$. We omit the subscript ξ_t to emphasize that the result can hold with different perturbations for any $t \in [n]$ and both when data is encountered newly or used for replay. In particular, the result only depends on the value of $\|\xi\|_2^2$, thus allowing arbitrary settings of ξ to be used for new or replayed data. From the formulation of the forward pass in (19), it can be shown that $f_{\mathbf{W}'}(x_t + \xi) = \sqrt{m} \cdot W_L' h'_{(t,L-1,\xi)}$ and $f_{\mathbf{W}}(x_t + \xi) = \sqrt{m} \cdot W_L h_{(t,L-1,\xi)}$. These identities allow us to derive the following via direct calculation

$$\begin{aligned}
&f_{\mathbf{W}'}(x_t + \xi) - F_{\mathbf{W},\mathbf{W}'}(x_t + \xi) \\
&= -\sqrt{m} \cdot \sum_{l=1}^{L-1} W_L \left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)} W_r\right) D_{(t,l,\xi)} \left(W_l' - W_l\right) h_{(t,l-1,\xi)} \\
&\quad + \sqrt{m} \cdot W_L' \left(h'_{(t,L-1,\xi)} - h_{(t,L-1,\xi)}\right)
\end{aligned} \tag{51}$$

Then, from Claim in 11.2 in [9], it is known that, for all $t \in [n]$, $h_{(t,L-1,\xi)} - h'_{(t,L-1,\xi)}$ (i.e., the red term within the above expression) can be re-written as

$$h_{(t,L-1,\xi)} - h'_{(t,L-1,\xi)} = \sum_{l=1}^{L-1} \left(\prod_{r=l+1}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W'_r \right) \left(D'_{(t,l,\xi)} + D''_{(t,l)} \right) \left(W_l - W'_l \right) h_{(t,l-1,\xi)}$$

where $D''_{(t,l)} \in \mathbb{R}^{m\times m}$ for $l \in [L-1]$ is any random diagonal matrix with at most $\mathcal{O}\left(m\omega^{\frac{2}{3}}L\right)$ non-zero entries in the range $[-1,1]$. With this in mind, we can then rewrite (51) as follows.

$$\begin{aligned}
&f_{\mathbf{W}'}(x_t+\xi) - F_{\mathbf{W},\mathbf{W}'}(x_t+\xi) \\
&= \sqrt{m}\cdot \sum_{l=1}^{L-1} W'_L \left(\prod_{r=l+1}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W'_r \right) \left(D'_{(t,l,\xi)} + D''_{(t,l)} \right) \left(W_l - W'_l \right) h_{(t,l-1,\xi)} \\
&\quad - \sqrt{m}\cdot \sum_{l=1}^{L-1} W_L \left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)} W_r \right) D_{(t,l,\xi)} \left(W'_l - W_l \right) h_{(t,l-1,\xi)}
\end{aligned}$$

Now, given that $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$, we can unroll this expression to arrive at the final result as follows

$$\begin{aligned}
&\left| f_{\mathbf{W}'}(x_t+\xi) - F_{\mathbf{W},\mathbf{W}'}(x_t+\xi) \right| \\
&= \sqrt{m}\cdot \left| \sum_{l=1}^{L-1} \left(W'_L \left(\prod_{r=l+1}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W'_r \right) \left(D'_{(t,l,\xi)} + D''_{(t,l)} \right) \left(W_l - W'_l \right) h_{(t,l-1,\xi)} \right) \right. \\
&\qquad \left. - \left(W_L \left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)} W_r \right) D_{(t,l,\xi)} \left(W'_l - W_l \right) h_{(t,l-1,\xi)} \right) \right| \\
&\overset{i}{\leq} \sqrt{m}\cdot \sum_{l=1}^{L-1} \left\| \left(W'_L \left(\prod_{r=l+1}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W'_r \right) \left(D'_{(t,l,\xi)} + D''_{(t,l)} \right) \right. \right. \\
&\qquad \left. + W_L \left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)} W_r \right) D_{(t,l,\xi)} \right\|_2 \cdot \|W_l - W'_l\|_2 \cdot \|h_{(t,l-1,\xi)}\|_2 \\
&\overset{ii}{\leq} \mathcal{O}\left(\sqrt{m}\omega^{\frac{1}{3}}L^2\log(m)\right) \sum_{l=1}^{L-1} \|W_l - W'_l\|_2 \cdot \|h_{(t,l-1,\xi)}\|_2 \\
&\overset{iii}{\leq} \mathcal{O}\left(\sqrt{m\left(1+\|\xi\|_2^2\right)}\omega^{\frac{1}{3}}L^2\log(m)\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)}+1\right) \right) \sum_{l=1}^{L-1} \|W_l - W'_l\|_2 \\
&\overset{iv}{\leq} \mathcal{O}\left(\sqrt{m\left(1+\|\xi\|_2^2\right)}\omega^{\frac{1}{3}}L^2\log(m)\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)}+1\right)(\omega L) \right) \\
&\overset{v}{\leq} \mathcal{O}\left(\sqrt{m\left(1+\|\xi\|_2^2\right)}\omega^{\frac{4}{3}}L^3\log(m) \right)
\end{aligned}$$

where i holds due to triangle inequality, ii holds due to Lemma 14 with probability at least $1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$, iii holds due to Corollary 2 with probability at least $1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$, iv holds due to the definition of the ω neighborhood, and v holds for sufficiently small ω. $\square$

D.5.5 Proof of Corollary 1

Proof. Consider some fixed $t \in [n]$ and arbitrary perturbation vector ξ, where we again omit the subscript ξ_t to emphasize that the result holds with different perturbations for any $t \in [n]$ and both when data is encountered newly or sampled for replay. We consider $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, where $\mathbf{W}^{(0)}$ is initialized as described within Section 5.5.1. Recall that we utilize a standard cross-entropy loss function $\ell(z) = \log(1 + e^{-z})$. We denote the derivative of this function as $\ell'(z)$, where $\ell'(z) = \frac{d}{dz}\log(1 + e^{-z}) = \frac{-1}{e^z+1}$. For $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, we can derive the following

$$\begin{aligned}
L_{(t,\xi)}(\mathbf{W}') - L_{(t,\xi)}(\mathbf{W}) &= \ell\left(y_t f_{\mathbf{W}'}(x_t + \xi)\right) - \ell\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \\
&\overset{i}{\geq} \ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \cdot \left(f_{\mathbf{W}'}(x_t + \xi) - f_{\mathbf{W}}(x_t + \xi)\right)
\end{aligned}$$

where i holds due to the convexity of $\ell(\cdot)$. From here, we have

$$\begin{aligned}
&\ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \cdot \left(f_{\mathbf{W}'}(x_t + \xi) - f_{\mathbf{W}}(x_t + \xi)\right) \\
&= \ell'(y_t f_{\mathbf{W}}(x_t + \xi)) \cdot y_t \cdot \Big(f_{\mathbf{W}'}(x_t + \xi) - f_{\mathbf{W}}(x_t + \xi) \\
&\qquad \pm \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t + \xi), \mathbf{W}' - \mathbf{W} \right\rangle\Big) \\
&= \ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \cdot \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t + \xi), \mathbf{W}' - \mathbf{W} \right\rangle \\
&\qquad + \ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \cdot \Big(f_{\mathbf{W}'}(x_t + \xi) - f_{\mathbf{W}}(x_t + \xi) \\
&\qquad - \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t + \xi), \mathbf{W}' - \mathbf{W} \right\rangle\Big) \\
&\geq \ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \cdot \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t + \xi), \mathbf{W}' - \mathbf{W} \right\rangle \\
&\qquad - \Big|\ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \cdot \Big(f_{\mathbf{W}'}(x_t + \xi) - f_{\mathbf{W}}(x_t + \xi) \\
&\qquad - \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t + \xi), \mathbf{W}' - \mathbf{W} \right\rangle\Big)\Big| \\
&= \sum_{l=1}^{L} \langle \nabla_{W_l} L_{(t,\xi)}(\mathbf{W}), W_l' - W_l \rangle - \Big|\ell'\left(y_t f_{\mathbf{W}}(x_t + \xi)\right) \cdot y_t \\
&\qquad \cdot \Big(f_{\mathbf{W}'}(x_t + \xi) - f_{\mathbf{W}}(x_t + \xi) - \left\langle \nabla_{\mathbf{W}} f_{\mathbf{W}}(x_t + \xi), \mathbf{W}' - \mathbf{W} \right\rangle\Big)\Big|
\end{aligned}$$

Then, by noticing that $|\ell'(y_t f_{\mathbf{W}}(x_t + \xi)) \cdot y_t)| \leq 1$ and invoking Lemma 12, we can derive the following with probability at least $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$

$$L_{(t,\xi)}(\mathbf{W}') - L_{(t,\xi)}(\mathbf{W}) \geq \sum_{l=1}^{L} \langle \nabla_{W_l} L_{(t,\xi)}(\mathbf{W}), W_l' - W_l \rangle - \mathcal{O}\left(\sqrt{m(1 + \|\xi\|_2^2)}\omega^{\frac{4}{3}} L^3 \log(m)\right)$$

whenever $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$. $\square$

D.5.6 Proof of Lemma 13

Proof. Consider some $t \in [n]$ and arbitrary perturbation vector ξ, where we omit the subscript ξ_t to emphasize that the result holds with different perturbations for any $t \in [n]$ and both when data is newly encountered or sampled for replay. Consider random weight matrices $\mathbf{W}^{(0)}$ initialized as in Section 5.5.1, and $\mathbf{W}$ such that $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$.

From Lemma 15, we have with probability at least $1 - \mathcal{O}\left(nL\right) e^{-\Omega(m/L)}$ that $\|h^{(0)}_{(t,j,\xi)}\|_2^2, \|g^{(0)}_{(t,j,\xi)}\|_2^2 \in \left[1 - \|\xi\|_2^2, 1 + \|\xi\|_2^2\right]$. Furthermore, if $m \geq \Omega(nL\log(nL))$, we have that for all $i \in [n]$ and all $1 \leq a \leq b \leq L$

$$\left\|\prod_{l=a}^{b} W_l^{(0)} D^{(0)}_{(t,l-1,\xi)}\right\|_2 \leq \mathcal{O}(\sqrt{L}) \tag{52}$$

with probability at least $1 - e^{-\Omega(m/L)}$ by Lemma 7.3 in [9].[50] Now, we can prove the desired result with an induction argument over layers of the network.

Base Case. $\|g_{(t,0,\xi)} - g^{(0)}_{(t,0,\xi)}\|_2 = \|x_t + \xi - (x_t + \xi)\|_2 = 0$, so the base case trivially holds. The same is true for $\|h_{(t,0,\xi)} - h^{(0)}_{(t,0,\xi)}\|_2$.

Inductive Case. Assume the inductive hypothesis holds for $l-1$. We can derive the following

$$\begin{aligned}
g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)} &= \left(W_l^{(0)} + W_l - W_l^{(0)}\right)\left(D^{(0)}_{(t,l-1,\xi)} + D_{(t,l-1,\xi)} - D^{(0)}_{(t,l-1,\xi)}\right)\left(g^{(0)}_{(t,l-1,\xi)} + g_{(t,l-1,\xi)} - g^{(0)}_{(t,l-1,\xi)}\right) \\
&\quad - W_l^{(0)} D^{(0)}_{(t,l-1,\xi)} g^{(0)}_{(t,l-1,\xi)} \\
&= \left(W_l - W_l^{(0)}\right)\left(D^{(0)}_{(t,l-1,\xi)} + D_{(t,l-1,\xi)} - D^{(0)}_{(t,l-1,\xi)}\right)\left(g^{(0)}_{(t,l-1,\xi)} + g_{(t,l-1,\xi)} - g^{(0)}_{(t,l-1,\xi)}\right) \\
&\quad + W_l^{(0)}\left(D_{(t,l-1,\xi)} - D^{(0)}_{(t,l-1,\xi)}\right)\left(g^{(0)}_{(t,l-1,\xi)} + g_{(t,l-1,\xi)} - g^{(0)}_{(t,l-1,\xi)}\right) \\
&\quad + W_l^{(0)} D^{(0)}_{(t,l-1,\xi)}\left(g_{(t,l-1,\xi)} - g^{(0)}_{(t,l-1,\xi)}\right)
\end{aligned}$$

From here, we can telescope over the $g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)}$ terms to obtain the expression below, where distinct terms of interest are highlighted with separate colors

$$\begin{aligned}
g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)} = \sum_{a=1}^{l}\left(\prod_{b=l}^{a+1} W_b^{(0)} D^{(0)}_{(t,b-1,\xi)}\right)\Bigg(\\
\left(W_a - W_a^{(0)}\right)\left(D^{(0)}_{(t,a-1,\xi)} + D_{(t,a-1,\xi)} - D^{(0)}_{(t,a-1,\xi)}\right)\left(g^{(0)}_{(t,a-1,\xi)} + g_{(t,a-1,\xi)} - g^{(0)}_{(t,a-1,\xi)}\right) \\
+ W_a^{(0)}\left(D_{(t,a-1,\xi)} - D^{(0)}_{(t,a-1,\xi)}\right)\left(g^{(0)}_{(t,a-1,\xi)} + g_{(t,a-1,\xi)} - g^{(0)}_{(t,a-1,\xi)}\right)\Bigg)
\end{aligned} \tag{53}$$

Now, we focus on a single summation element of the green term in (53) and provide an upper bound on the

[50](52) has one extra factor of $D^{(0)}_{(t,l-1,\xi)}$ within the expression in comparison to Lemma 7.3 in [9], but this does not impact the norm of the overall expression because $\left\|D^{(0)}_{l-1}\right\|_2 \leq 1$.

norm of this expression.

$$
\begin{aligned}
&\left\| \left(\prod_{b=l}^{a+1} W_b^{(0)} D_{(t,b-1,\xi)}^{(0)} \right) \left(W_a - W_a^{(0)} \right) \left(D_{(t,a-1,\xi)}^{(0)} + D_{(t,a-1,\xi)} - D_{(t,a-1,\xi)}^{(0)} \right) \right. \\
&\qquad \left. \left(g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right) \right\|_2 \\
&\leq \left\| \prod_{b=l}^{a+1} W_b^{(0)} D_{(t,b-1,\xi)}^{(0)} \right\|_2 \cdot \left\| W_a - W_a^{(0)} \right\|_2 \cdot \| D_{(t,a-1,\xi)} \|_2 \cdot \left\| g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right\|_2 \\
&\overset{i}{\leq} \mathcal{O}\left(\sqrt{L}\right) \cdot \left\| W_a - W_a^{(0)} \right\|_2 \cdot \| D_{(t,a-1,\xi)} \|_2 \cdot \left\| g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right\|_2 \\
&\overset{ii}{\leq} \mathcal{O}\left(\sqrt{L}\omega\right) \cdot \| D_{(t,a-1,\xi)} \|_2 \cdot \left\| g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right\|_2 \\
&\overset{iii}{\leq} \mathcal{O}\left(\sqrt{L}\omega\right) \cdot \left\| g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right\|_2 \\
&\overset{iv}{\leq} \mathcal{O}\left(\sqrt{L}\omega\right) \left[\| g_{(t,a-1,\xi)}^{(0)} \|_2 + \| g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \|_2 \right] \\
&\overset{v}{\leq} \mathcal{O}\left(\sqrt{L}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2) \log(m)} \right) \right]
\end{aligned}
$$

where i holds due to (52), ii holds because $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, iii holds because $\|D_{i,a-1}\|_2 \leq 1$ by construction, iv holds by upper triangle inequality, and v holds by Lemma 15 and the inductive hypothesis.

Now that we have bounded the green term, we can focus on a single summation element of the red term in (53). First, we make the following definition

$$
\begin{aligned}
\zeta &\triangleq \left(D_{(t,a-1,\xi)} - D_{(t,a-1,\xi)}^{(0)} \right) \left(g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right) \\
&= \left(D_{(t,a-1,\xi)} - D_{(t,a-1,\xi)}^{(0)} \right) \left(W_{a-1}^{(0)} h_{(t,a-2,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right)
\end{aligned}
$$

Then, by Claim 8.3 and Corollary 8.4 in [9], we have that with probability at least $1 - e^{-\Omega(m\omega^{\frac{2}{3}} L)}$

$$
\left\| \frac{1}{c} \zeta \right\|_0 \leq \mathcal{O}\left(m\omega^{\frac{2}{3}} L \right) \quad \text{and} \quad \left\| \frac{1}{c} \zeta \right\|_2 \leq \mathcal{O}\left(\omega L^{\frac{3}{2}} \right) \tag{54}
$$

where $c = \|h_{(t,a-2,\xi)}^{(0)}\|_2 \in [\sqrt{1 - \|\xi\|_2^2}, \sqrt{1 + \|\xi\|_2^2}]$. Additionally, we define

$$
\begin{aligned}
\gamma &\triangleq \left(\prod_{b=l}^{a+1} W_b^{(0)} D_{(t,b-1,\xi)} \right) \cdot W_a^{(0)} \cdot \left[\left(D_{(t,a-1,\xi)} - D_{(t,a-1,\xi)}^{(0)} \right) \left(g_{(t,a-1,\xi)}^{(0)} + g_{(t,a-1,\xi)} - g_{(t,a-1,\xi)}^{(0)} \right) \right] \\
&= c \left(\prod_{b=l}^{a+1} W_b^{(0)} D_{(t,b-1,\xi)} \right) \cdot W_a^{(0)} \cdot \left[\frac{1}{c} \zeta \right]
\end{aligned}
$$

By invoking Claim 8.5 from [9], we can reformulate γ as

$$\gamma = \left(c \left\| \frac{1}{c}\zeta \right\|_2 \right) (\gamma_1 + \gamma_2)$$

where with probability at least $1 - e^{-\Omega(m\omega^{\frac{2}{3}} L \log(m))}$ the γ_1 and γ_2 terms can be bounded as

$$\|\gamma_1\|_2 \leq \mathcal{O}\left(L^{\frac{1}{2}}\omega^{\frac{1}{3}}\log(m)\right) \text{ and } \|\gamma_2\|_\infty \leq \mathcal{O}\left(\sqrt{\frac{\log(m)}{m}}\right) \tag{55}$$

Combining all of this together, we get

$$\begin{aligned}
\|g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)}\|_2 &= \left\| \sum^{l} \Bigg(\mathcal{O}\left(\sqrt{L}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \right] \right. \\
&\qquad \left. + \left(c \left\| \frac{1}{c}\zeta \right\|_2 \right)(\gamma_1 + \gamma_2) \Bigg) \right\|_2 \\
&= \left\| L \Bigg(\mathcal{O}\left(\sqrt{L}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \right] \right. \\
&\qquad \left. + \left(c \left\| \frac{1}{c}\zeta \right\|_2 \right)(\gamma_1 + \gamma_2) \Bigg) \right\|_2 \\
&\overset{i}{\leq} \mathcal{O}\left(L^{\frac{3}{2}}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \right] \\
&\qquad + L\left(c \left\| \frac{1}{c}\zeta \right\|_2 \right) \|\gamma_1 + \gamma_2\|_2 \\
&\overset{ii}{\leq} \mathcal{O}\left(L^{\frac{3}{2}}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \right] \\
&\qquad + \left(\sqrt{1 + \|\xi\|_2^2}\, \mathcal{O}\left(\omega L^{\frac{5}{2}}\right) \right) \|\gamma_1 + \gamma_2\|_2 \\
&\overset{iii}{\leq} \mathcal{O}\left(L^{\frac{3}{2}}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \right] \\
&\qquad + \left(\sqrt{1 + \|\xi\|_2^2}\, \mathcal{O}\left(\omega L^{\frac{5}{2}}\right) \right) (\|\gamma_1\|_2 + \|\gamma_2\|_2) \\
&\overset{iv}{\leq} \mathcal{O}\left(L^{\frac{3}{2}}\omega\right) \left[\sqrt{1 + \|\xi\|_2^2} + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \right] \\
&\qquad + \left(\sqrt{1 + \|\xi\|_2^2}\, \mathcal{O}\left(\omega L^{\frac{5}{2}}\right) \right) \left(\mathcal{O}\left(L^{\frac{1}{2}}\omega^{\frac{1}{3}}\log(m)\right) + \mathcal{O}\left(\sqrt{\log(m)}\right) \right) \\
&= \mathcal{O}\left(L^{\frac{3}{2}}\omega\sqrt{1 + \|\xi\|_2^2}\right) + \mathcal{O}\left(\omega^2 L^4 \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right) \\
&\qquad \mathcal{O}\left(\omega^{\frac{4}{3}} L^3 \sqrt{1 + \|\xi\|_2^2}\log(m)\right) + \mathcal{O}\left(\omega L^{\frac{5}{2}} \sqrt{(1 + \|\xi\|_2^2)\log(m)}\right)
\end{aligned}$$

where i holds by the upper triangle inequality and properties of norms, ii holds by Lemma 15 and (54), iii

holds by triangle inequality, and iv holds by (55) and invoking the upper bound $\|\gamma_2\|_2 \leq \sqrt{m}\|\gamma_2\|_\infty$. Then, when ω is sufficiently small, we get

$$\begin{aligned}\|g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)}\|_2 &\leq \mathcal{O}\left(L^{\frac{3}{2}}\omega\sqrt{1+\|\xi\|_2^2} + \omega L^{\frac{5}{2}}\sqrt{1+\|\xi\|_2^2}\sqrt{\log(m)}\right) \\ &\leq \mathcal{O}\left(\omega L^{\frac{5}{2}}\sqrt{\left(1+\|\xi\|_2^2\right)\log(m)}\right) \qquad (56)\end{aligned}$$

thus completing the inductive portion of the proof for $\|g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)}\|_2$. Then, to finish the inductive portion of the proof for $\|h_{(t,l,\xi)} - h^{(0)}_{(t,l,\xi)}\|_2$, we note that

$$\begin{aligned}\left\|h_{(t,l,\xi)} - h^{(0)}_{(t,l,\xi)}\right\|_2 &= \|D_{(t,l,\xi)}\left(g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)}\right) + \left(D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right)g_{(t,l,\xi)}\|_2 \\ &\overset{i}{\leq} \|D_{(t,l,\xi)}\|_2 \cdot \left\|g_{(t,l,\xi)} - g^{(0)}_{(t,l,\xi)}\right\|_2 + \left\|D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right\|_2 \cdot \|g_{(t,l,\xi)}\|_2 \\ &\overset{ii}{\leq} 1 \cdot \mathcal{O}\left(\omega L^{\frac{5}{2}}\sqrt{1+\|\xi\|_2^2}\sqrt{\log(m)}\right) + 1 \cdot \left(\omega L^{\frac{3}{2}}\sqrt{1+\|\xi\|_2^2}\right) \\ &\leq \mathcal{O}\left(\omega L^{\frac{5}{2}}\sqrt{1+\|\xi\|_2^2}\sqrt{\log(m)}\right)\end{aligned}$$

where i holds from the upper triangle inequality and properties of the spectral norm and ii holds from (54) and (56). Thus, we have completed the inductive case for $\|h_{(t,l,\xi)} - h^{(0)}_{(t,l,\xi)}\|_2$. From here, a union bound can be taken over all $t \in [n]$ and $l \in [L-1]$ to yield the desired result with probability $1 - \mathcal{O}(nL)e^{-\Omega(m/L)} - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)} + \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\log(m)\right)} = 1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$, where the last equality again holds when ω is sufficiently small. $\square$

D.5.7 Proof of Lemma 14

Proof. We consider some fixed $t \in [n]$ and arbitrary perturbation vector ξ, where we omit the subscript ξ_t to emphasize that the result holds with different perturbations for any $t \in [n]$ and both when data is newly encountered or sampled for replay. We define random diagonal matrices $D''_{(t,1)}, \dots, D''_{(t,L-1)}$ as any diagonal matrix with at most $\mathcal{O}\left(m\omega^{\frac{2}{3}}L\right)$ entries in the range $\in [-1,1]$. Consider two sets of model parameters $\mathbf{W}, \mathbf{W}' \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$, where $\mathbf{W}^{(0)}$ is initialized as described in Section 5.5.1. From (54), we know that

$$\left\|\frac{1}{\|h^{(0)}_{(t,l-1,\xi)}\|_2}\left(D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right)g_{(t,l,\xi)}\right\|_0 \leq \mathcal{O}\left(m\omega^{\frac{2}{3}}L\right)$$

with probability at least $1 - e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. This bound holds for both $\mathbf{W}$ and $\mathbf{W}'$. Then, noticing that right multiplication by $g_{(t,l,\xi)}$ and division by a constant factor cannot increase the number of non-zero entries within the matrix $D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}$ (i.e., recall that this matrix is diagonal), we have

$$\left\|D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right\|_0 \leq \mathcal{O}\left(m\omega^{\frac{2}{3}}L\right) \qquad (57)$$

From here, we can apply the upper triangle inequality to yield

$$\begin{aligned}\left\|D_{(t,l,\xi)} - D'_{(t,l,\xi)}\right\|_0 &= \left\|\left(D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right) - \left(D'_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right)\right\|_0 \\ &\overset{i}{\leq} \left\|D_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right\|_0 + \left\|D'_{(t,l,\xi)} - D^{(0)}_{(t,l,\xi)}\right\|_0 \\ &\overset{ii}{=} \mathcal{O}\left(m\omega^{\frac{2}{3}}L\right)\end{aligned}$$

where i holds by the upper triangle inequality and ii is due to (57). From here, we apply a union bound over all $t \in [n]$ and $l \in [L-1]$ to yield

$$\left\|D_{(t,l,\xi)} - D'_{(t,l,\xi)}\right\|_0 \leq \mathcal{O}\left(m\omega^{\frac{2}{3}}L\right) \tag{58}$$

for all $t \in [n]$ and $l \in [L-1]$ with probability at least $1 - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. (58) can also be extended to show the following properties with identical probability

$$\begin{aligned}\left\|D_{(t,l,\xi)} + D''_{(t,l)} - D^{(0)}_{(t,l,\xi)}\right\|_0 \leq \mathcal{O}\left(m\omega^{\frac{2}{3}}L\right) \\ \left\|D'_{(t,l,\xi)} + D''_{(t,l)} - D^{(0)}_{(t,l,\xi)}\right\|_0 \leq \mathcal{O}\left(m\omega^{\frac{2}{3}}L\right)\end{aligned} \tag{59}$$

which holds due to the number of non-zero entries assumed to be within each random diagonal matrix $D''_{(t,l)}$ by construction. From here, we first note that with probability at least $1 - e^{-\Omega\left(m\omega^{\frac{2}{3}}L\log(m)\right)}$ we have

$$\left\|\prod_{r=l}^{L-1}\left(D_{(t,l,\xi)} - D''_{(t,l)}\right)W_l\right\|_2 \leq \mathcal{O}\left(\sqrt{L}\right) \tag{60}$$

due to Lemma 8.6 in [9]. Then, we consider bounding the following expression

$$
\begin{aligned}
&\left\| W_L' \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' - W_L \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \right\|_2 \\
&= \Bigg\| \left(W_L' + W_L^{(0)} - W_L^{(0)} \right) \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' \\
&\qquad - \left(W_L - W_L^{(0)} + W_L^{(0)} \right) \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \Bigg\|_2 \\
&= \Bigg\| \left(W_L' - W_L^{(0)} \right) \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' + W_L^{(0)} \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' \\
&\qquad - \left(W_L - W_L^{(0)} \right) \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r - W_L^{(0)} \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \Bigg\|_2 \\
&\overset{i}{\le} \left\| \left(W_L' - W_L^{(0)} \right) \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' \right\|_2 + \left\| \left(W_L - W_L^{(0)} \right) \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \right\|_2 \\
&\qquad + \left\| W_L^{(0)} \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' - W_L^{(0)} \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \right\|_2 \\
&\overset{ii}{\le} \mathcal{O}\left(\sqrt{L}\omega \right) + \left\| W_L^{(0)} \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' - W_L^{(0)} \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \right\|_2 \\
&= \mathcal{O}\left(\sqrt{L}\omega \right) + \Bigg\| W_L^{(0)} \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' - W_L^{(0)} \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r \\
&\qquad \pm W_L^{(0)} \prod_{r=l}^{L-1} D^{(0)}_{(t,r,\xi)} W_r^{(0)} \Bigg\|_2 \\
&\overset{iii}{\le} \mathcal{O}\left(\sqrt{L}\omega \right) + \left\| W_L^{(0)} \prod_{r=l}^{L-1} \left(D'_{(t,r,\xi)} + D''_{(t,r)} \right) W_r' - W_L^{(0)} \prod_{r=l}^{L-1} D^{(0)}_{(t,r,\xi)} W_r^{(0)} \right\|_2 \\
&\qquad + \left\| W_L^{(0)} \prod_{r=l}^{L-1} D_{(t,r,\xi)} W_r - W_L^{(0)} \prod_{r=l}^{L-1} D^{(0)}_{(t,r,\xi)} W_r^{(0)} \right\|_2 \\
&\overset{iv}{\le} \mathcal{O}\left(\sqrt{L}\omega \right) + \mathcal{O}\left(\omega^{\frac{1}{3}} L^2 \sqrt{\log(m)} \right) \\
&\overset{v}{\le} \mathcal{O}\left(\omega^{\frac{1}{3}} L^2 \sqrt{\log(m)} \right)
\end{aligned}
$$

where i and iii hold due to the upper triangle inequality, ii holds due to (60), iv holds by Lemma 8.7 in [9] with probability at least $1 - e^{-\Omega\left(m\omega^{\frac{2}{3}} L \log(m)\right)}$, and v holds for $\omega \le \mathcal{O}\left(L^{-6} \log^{-3}(m)\right)$. Then, the desired result follows with probability at least $1 - \mathcal{O}(nL) e^{-\Omega\left(m\omega^{\frac{2}{3}} L\right)} - \mathcal{O}(nL) e^{-\Omega\left(m\omega^{\frac{2}{3}} L \log(m)\right)} = 1 - \mathcal{O}(nL) e^{-\Omega\left(m\omega^{\frac{2}{3}} L\right)}$ by taking a union bound across all $t \in [n]$ and $l \in [L-1]$. $\square$

D.5.8 Proof of Lemma 15

Proof. Consider some fixed $t \in [n]$, $j \in [L-1]$, and arbitrary perturbation vector ξ, where we omit the subscript ξ_t to emphasize that the result holds with different perturbations for any $t \in [n]$ and both when data is newly encountered or sampled for replay. Assume the neural network weights at initialization $\mathbf{W}^{(0)}$ follow the initialization scheme describe in Section 5.5.1. For $l \in [L]$, we can define $\Delta_l^{(0)} \triangleq \frac{\|h^{(0)}_{(t,l,\xi)}\|_2^2}{\|h^{(0)}_{(t,l-1,\xi)}\|_2^2}$. Applying a logarithm (with arbitrary base $a > 1$) to this definition, we have

$$\log\left(\left\|h^{(0)}_{(t,j,\xi)}\right\|_2^2\right) = \log(\|x_t + \xi\|_2^2) + \sum_{l=0}^{j} \log\left(\Delta_l^{(0)}\right)$$

Given some $\epsilon \in (0, 1]$, we can invoke Fact 7.2 and the proof of Lemma 7.1 from [9] to show that

$$\left|\sum_{l=0}^{j} \log\left(\Delta_l^{(0)}\right)\right| \leq \epsilon$$

with probability at least $1 - \mathcal{O}(e^{-\Omega(\epsilon^2 m/L)})$. Thus, we have

$$\log(\|x_t + \xi\|_2^2) - \epsilon \leq \log\left(\left\|h^{(0)}_{(t,j,\xi)}\right\|_2^2\right) \leq \log(\|x_t + \xi\|_2^2) + \epsilon \tag{61}$$

We first expand the upper bound to derive a bound on $\|h^{(0)}_{(t,j,\xi)}\|_2^2$. We begin by exponentiating both sides of the inequality in (61) to obtain the following

$$\begin{aligned}
\left\|h^{(0)}_{(t,j,\xi)}\right\|_2^2 &\leq a^{\log(\|x_t+\xi\|_2^2)+\epsilon} \\
&= (\|x_t + \xi\|_2^2) \cdot a^\epsilon \\
&\overset{i}{\leq} (\|x_t\|_2^2 + \|\xi\|_2^2) \cdot a^\epsilon \\
&\overset{ii}{\leq} (\|x_t\|_2^2 + \|\xi\|_2^2) \cdot a \\
&\overset{iii}{=} (1 + \|\xi\|_2^2) \cdot a
\end{aligned}$$

where i follows from the upper triangle inequality, ii is implied by the fact that $a > 1$ and $\epsilon \in (0, 1]$, and iii follows from the unit norm assumption on input data. From here, we note that the base a chosen for the logarithm is arbitrary, and that any base greater than one can be chosen. With this in mind, we note that $\lim_{a \to 1^+}(1 + \|\xi\|_2^2) \cdot a = 1 + \|\xi\|_2^2$, which yields the upper bound $\|h^{(0)}_{(t,j,\xi)}\|_2^2 \leq 1 + \|\xi\|_2^2$.

We can similarly derive a lower bound on $\|h^{(0)}_{(t,j,\xi)}\|_2^2$ as follows, where we begin by exponentiating both sides of (61)

$$\begin{aligned}\|h^{(0)}_{(t,j,\xi)}\|_2^2 &\geq a^{\log(\|x_t+\xi\|_2^2)-\epsilon}\\ &= (\|x_t+\xi\|_2^2)\cdot a^{-\epsilon}\\ &\overset{i}{\geq} \left(\left|\|x_t\|_2^2-\|\xi\|_2^2\right|\right)\cdot a^{-\epsilon}\\ &\overset{ii}{\geq} (1-\|\xi\|_2^2)\cdot a^{-\epsilon}\\ &\overset{iii}{\geq} (1-\|\xi\|_2^2)\cdot \frac{1}{a}\end{aligned}$$

where i follows from the lower triangle inequality, ii follows from the norm assumption on the data, and iii follows from the fact that $\epsilon \in (0,1]$ and $a > 1$. Noting that the base a chosen for the logarithm is arbitrary, we have $\lim_{a\to 1^+}(1-\|\xi\|_2^2)\cdot \frac{1}{a} = 1-\|\xi\|_2^2$.

By invoking both the upper and lower bounds derived above, we end up with $\|h^{(0)}_{(t,j,\xi)}\|_2^2 \in \left[1-\|\xi\|_2^2, 1+\|\xi\|_2^2\right]$ with probability at least $1-\mathcal{O}\left(e^{-\Omega(m/L)}\right)$ due to the fact that $\epsilon \in (0,1]$. From here, we can take a union bound over all all $t \in [n]$ and $j \in [L-1]$ to yield the final result with probability $1-\mathcal{O}\left(nL\right)e^{-\Omega(m/L)}$. □

D.5.9 Proof of Corollary 2

Proof. Consider some fixed $t \in [n]$, $j \in [L-1]$, and arbitrary perturbation vector ξ, where we omit the subscript ξ_t to emphasize that the result holds with different perturbations for any $t \in [n]$ and both when data is newly encountered or sampled for replay. Assume the neural network weights at initialization $\mathbf{W}^{(0)}$ follow the initialization scheme described in Section 5.5.1. From here, we take $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$. If we then assume $\omega \leq \mathcal{O}\left(L^{-\frac{9}{2}}\log^{-3}(m)\right)$, then we have from Lemma 13 that

$$\left\|h_{(t,j,\xi)} - h^{(0)}_{(t,j,\xi)}\right\|_2 \leq \mathcal{O}\left(\omega L^{\frac{5}{2}}\sqrt{(1+\|\xi\|_2^2)\log(m)}\right) \tag{62}$$

with probability at least $1-e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. Similarly, from Lemma 15, we have the following

$$\left\|h^{(0)}_{(t,j,\xi)}\right\|_2 \leq \sqrt{1+\|\xi\|_2^2}$$

with probability at least $1-e^{-\Omega(m/L)}$. Then, we can use these expressions to derive a bound on $\|h_{(t,j,\xi)}\|_2$ as follows

$$\begin{aligned}\left\|h_{(t,j,\xi)} - h^{(0)}_{(t,j,\xi)}\right\|_2 &\overset{i}{\geq} \left|\|h_{(t,j,\xi)}\|_2 - \left\|h^{(0)}_{(t,j,\xi)}\right\|_2\right|\\ &\overset{ii}{\geq} \left|\|h_{(t,j,\xi)}\|_2 - \sqrt{1+\|\xi\|_2^2}\right|\\ &\geq \|h_{(t,j,\xi)}\|_2 - \sqrt{1+\|\xi\|_2^2}\end{aligned}$$

where i follows from the lower triangle inequality and ii follows from Lemma 15. Then, combining the expression above with (62), we derive

$$\begin{aligned}\|h_{(t,j,\xi)}\|_2 &\leq \mathcal{O}\left(\omega L^{\frac{5}{2}}\sqrt{(1+\|\xi\|_2^2)\log(m)}\right) + \sqrt{1+\|\xi\|_2^2} \\ &= \mathcal{O}\left(\sqrt{1+\|\xi\|_2^2}\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)}+1\right)\right)\end{aligned}$$

Then, by taking a union bound over all $t \in [n]$ and $j \in [L-1]$, we have the desired result with probability at least $1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. □

D.5.10 Proof of Lemma 16

Proof. Consider some fixed $t \in [n]$, $l \in [L-1]$, and arbitrary perturbation vector ξ, where we omit the subscript ξ_t to emphasize that the result holds with different perturbations for any $t \in [n]$ and both when data is newly encountered or sampled for replay. Given $\mathbf{W} \in \mathcal{B}(\mathbf{W}^{(0)}, \omega)$ with $\mathbf{W}^{(0)}$ initialized as described in Section 5.5.1, we have

$$\begin{aligned}\|\nabla_{W_L} f_{\mathbf{W}}(x_t+\xi)\|_2 &\overset{i}{=} \left\|\sqrt{m}\cdot h_{(t,L-1,\xi)}\right\|_2 \\ &\overset{ii}{\leq} \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)}+1\right)\right)\end{aligned}$$

where i holds because $\nabla_{W_L} f_{\mathbf{W}}(x_t+\xi) = \sqrt{m}\cdot h_{(t,L-1,\xi)}^{\top}$ and ii holds due to Corollary 2 with probability at least $1-e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. For layers $l \in [L-1]$, we have

$$\nabla_{W_l} f_{\mathbf{W}}(x_t+\xi) = \sqrt{m}\cdot\left(h_{(t,l-1,\xi)}W_L\left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)}W_r\right)D_{(t,l,\xi)}\right)^{\top}$$

which allows us to show

$$\begin{aligned}\|\nabla_{W_l} f_{\mathbf{W}}(x_t+\xi)\|_F &= \sqrt{m}\cdot\left\|h_{(t,l-1,\xi)}W_L\left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)}W_r\right)D_{(t,l,\xi)}\right\|_F \\ &\overset{i}{=} \sqrt{m}\cdot\|h_{(t,l-1,\xi)}\|_2\cdot\left\|W_L\left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)}W_r\right)\right\|_2 \\ &\overset{ii}{\leq} \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)}+1\right)\right) \\ &\quad \cdot\left\|W_L\left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)}W_r\right)\right\|_2\end{aligned} \tag{63}$$

where i holds due to properties of norms and ii holds due to Corollary 2 with probability at least $1-e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. Now, we must bound the red term within the expression above to complete the proof

$$
\begin{aligned}
\left\|W_L\left(\prod_{r=l+1}^{L-1} D_{(t,r,\xi)} W_r\right)\right\|_2 &\overset{i}{\leq} \left\|W_L\left(\prod_{r=l+1}^{L-1}\left(D_{(t,r,\xi)}+D''_{(t,r)}\right) W_r\right)\right\|_2 \\
&= \left\|W_L\left(\prod_{r=l+1}^{L-1}\left(D_{(t,r,\xi)}+D''_{(t,r)}\right) W_r\right)\right. \\
&\qquad \left. \pm W_L^{(0)} \prod_{r=l+1}^{L-1} D^{(0)}_{(t,r,\xi)} W_r^{(0)}\right\|_2 \\
&\overset{ii}{\leq} \left\|W_L\left(\prod_{r=l+1}^{L-1}\left(D_{(t,r,\xi)}+D''_{(t,r)}\right) W_r\right)\right. \\
&\qquad \left. - W_L^{(0)} \prod_{r=l+1}^{L-1} D^{(0)}_{(t,r,\xi)} W_r^{(0)}\right\|_2 \\
&\qquad + \left\|W_L^{(0)} \prod_{r=l+1}^{L-1} D^{(0)}_{(t,r,\xi)} W_r^{(0)}\right\|_2 \\
&\overset{iii}{\leq} \mathcal{O}\left(\omega^{\frac{1}{3}} L^2 \sqrt{\log(m)}\right) + \mathcal{O}(1) \\
&= \mathcal{O}(1)
\end{aligned}
\tag{64}
$$

where i holds for some random diagonal matrix $D''_{(t,l)} \in [-1,1]^{m\times m}$ with at most $\mathcal{O}\left(m\omega^{\frac{2}{3}} L\right)$ non-zero entries and ii is due to the upper triangle inequality. iii holds due to Lemma 14 when $\omega \leq \mathcal{O}\left(L^{-6}\log^{-3}(m)\right)$ and Lemma 7.4 in [9] with probabilities at least $1-e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$ and $1-e^{-\Omega(m/L)}$, respectively. Thus, we have from (63) and (64)

$$
\begin{aligned}
\|\nabla_{W_l} f_{\mathbf{W}}(x_t+\xi)\|_F &\leq \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\left(\omega L^{\frac{5}{2}}\sqrt{\log(m)}+1\right)\right) \\
&\leq \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\right)
\end{aligned}
$$

where the final inequality holds given sufficiently small ω. Then, a union bound can be taken over all $t \in [n]$ and $l \in [L-1]$ to yield the desired bound on $\|\nabla_{W_l} f_{\mathbf{W}}(x_t+\xi)\|_F$ with probability at least $1-\mathcal{O}(nL)e^{-\Omega(m/L)} - \mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)} = 1-\mathcal{O}(nL)e^{-\Omega\left(m\omega^{\frac{2}{3}}L\right)}$. From here, we translate this result to a bound on $\|\nabla_{W_l} L_{(t,\xi)}(\mathbf{W})\|_F$ as follows

$$
\begin{aligned}
\left\|\nabla_{W_l} L_{(t,\xi)}(\mathbf{W})\right\|_F &= \left\|\ell'\left(y_t \cdot f_{\mathbf{W}}(x_t+\xi)\right)\cdot y_t \cdot \nabla_{W_l} f_{\mathbf{W}}(x_t+\xi)\right\|_F \\
&= \left|\ell'\left(y_t \cdot f_{\mathbf{W}}(x_t+\xi)\right)\cdot y_t\right| \cdot \|\nabla_{W_l} f_{\mathbf{W}}(x_t+\xi)\|_F \\
&\leq \mathcal{O}\left(\sqrt{m(1+\|\xi\|_2^2)}\right)
\end{aligned}
$$

where the final inequality is derived by noticing that $|\ell' (y_t \cdot f_{\mathbf{W}}(x_t + \xi)) \cdot y_t| \leq 1$ and invoking the bound on $\|\nabla_{W_l} f_{\mathbf{W}}(x_t + \xi)\|_F$ derived above. Thus, we have arrived at the desired result. □

E Better Schedules for Low Precision Training of Deep Neural Networks

E.1 Supplementary Information

All code for this project is publicly-available via github at the following link: `https://github.com/wolfecameron/BetterPrecisionSchedules`