Incremental Learning via Rate Reduction

Kyung Eun Baek
Yi Ma

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-130
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-130.html

May 14, 2021

Incremental Learning via Rate Reduction

by Christina Baek

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Yi Ma

Professor Yi Ma
Research Advisor

May 13, 2021

(Date)

* * * * * * *

Anant Sahai

Professor Anant Sahai
Second Reader

May 14, 2021

(Date)

Incremental Learning via Rate Reduction

by

Christina Baek

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Yi Ma, Chair
Professor Anant Sahai

Spring 2021

Abstract

Incremental Learning via Rate Reduction

by

Christina Baek

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Yi Ma, Chair

Current deep learning architectures suffer from catastrophic forgetting, a failure to retain knowledge of previously learned classes when incrementally trained on new classes. The fundamental roadblock faced by deep learning methods is that the models are optimized as "black boxes," making it difficult to properly adjust the model parameters to preserve knowledge about previously seen data. To overcome the problem of catastrophic forgetting, we propose utilizing an alternative "white box" architecture derived from the principle of rate reduction, where each layer of the network is explicitly computed without back propagation. Under this paradigm, we demonstrate that, given a pretrained network and new data classes, our approach can provably construct a new network that emulates joint training with all past and new classes. Finally, our experiments show that our proposed learning algorithm observes significantly less decay in classification performance, outperforming state of the art methods on MNIST and CIFAR-10 by a large margin and justifying the use of "white box" algorithms for incremental learning even for sufficiently complex image data.

Contents

List of Figures

List of Tables

Acknowledgments

This work was a joint effort with students Ziyang Wu and Chong You, advised by Professor Yi Ma. I would like to thank Professor Yi Ma for the invaluable opportunities he has given me this year to learn and grow as a student and researcher. I learned tremendously from him and graduate students Simon, Yaodong, Chong, Xili, and others, whose dedication and compassion inspired me on a daily basis to keep working towards a better understanding of this field. I dedicate my thesis to my family, from whom I always receive unwavering support.

Chapter 1

Interpretable Network from Rate Reduction

1.1 Principle of Maximal Coding Rate Reduction

Given a set of training data $\{\boldsymbol{x}_i\}$ and their corresponding labels $\{\boldsymbol{y}_i\}$, classical deep learning aims to learn a nonlinear mapping $h(\cdot) : \boldsymbol{x} \to \boldsymbol{y}$, implemented as a series of simple linear and nonlinear maps, that minimizes the cross-entropy loss. One popular way to interpret the role of multiple layers is to consider the output of each intermediate layer as a latent representation space. Then, the beginning layers aim to learn a latent representation $\boldsymbol{z} = f(\boldsymbol{x}, \theta) \in \mathbb{R}^d$ that best facilitates the later layers $\boldsymbol{y} = g(\boldsymbol{z})$ for the downstream classification task.

$$\boldsymbol{x} \xrightarrow{f(\boldsymbol{x},\theta)} \boldsymbol{z}(\theta) \xrightarrow{g(\boldsymbol{z})} \boldsymbol{y}$$

As a concrete example, in image recognition tasks, $f(\cdot)$ is a convolutional backbone that encodes an image $\boldsymbol{x} \in \mathbb{R}^{H\times W\times C}$ into a vector representation $\boldsymbol{z} = f(\boldsymbol{x}, \theta) \in \mathbb{R}^d$ and $g(\boldsymbol{z}) = \boldsymbol{w} \cdot \boldsymbol{z}$ is a linear classifier where $\boldsymbol{w} \in \mathbb{R}^{k\times d}$ and k is the number of classes. Recent work [15] shows that this direct label fitting leads to a phenomena called *neural collapse* in deep networks, where within-class variability and structural information are completely suppressed. Namely, at the final hidden layer, the variance of each latent class distribution often converges to 0, such that for each class, the training inputs map to a single point. Therefore, it is unclear to what extent the feature representation captures any intrinsic structure of the data.

To address the aforementioned problem, a recent work by Yu *et al.* [25] presented a framework for learning useful and geometrically meaningful representations by maximizing the coding rate reduction (i.e., MCR^2). Given m training samples of d dimension $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m] \in \mathbb{R}^{d\times m}$ that belong to k classes, let $\boldsymbol{Z} = [f(\boldsymbol{x}_1, \theta), ..., f(\boldsymbol{x}_m, \theta)] \in \mathbb{R}^{d\times m}$ be the latent representation. Let $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}^j\}_{j=1}^k$ be the membership of the data in the k classes, where each $\boldsymbol{\Pi}^j \in \mathbb{R}^{m\times m}$ is a diagonal matrix such that $\boldsymbol{\Pi}^j(i, i)$ is the probability of $\boldsymbol{x}_i$ belonging to class j. Then, MCR^2 aims to learn a feature representation $\boldsymbol{Z}$ by maximizing

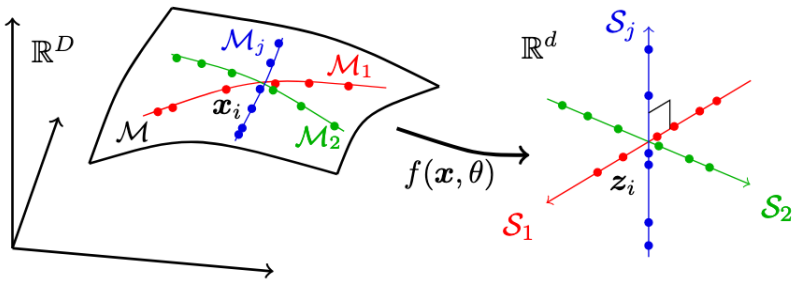


Figure 1.1: MCR^2 learns a map $f(\boldsymbol{x},\theta)$ such that the class representation $\boldsymbol{Z}_j = f(\boldsymbol{X}_j,\theta)$ lies on maximally uncorrelated subspaces $\{S_j\}$.

the following *rate reduction*:

$$\Delta R(\boldsymbol{Z}) = R(\boldsymbol{Z}) - R_c(\boldsymbol{Z},\boldsymbol{\Pi}), \tag{1.1}$$

where

$$R(\boldsymbol{Z}) = \frac{1}{2}\log\det\left(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^\top\right), \quad \text{and} \tag{1.2}$$

$$R_c(\boldsymbol{Z},\boldsymbol{\Pi}) = \sum_{j=1}^{k}\frac{\gamma_j}{2}\log\det\left(\boldsymbol{I} + \alpha_j \boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^\top\right). \tag{1.3}$$

subject to the constraint that $\boldsymbol{Z}$ is properly normalized, *i.e.*, have the Frobenius norm of class features $\boldsymbol{Z}^j = \boldsymbol{Z}\boldsymbol{\Pi}^j$ scale with the number of samples in class j: $\|\boldsymbol{Z}^j\|_F^2 = m_j = \mathrm{tr}(\boldsymbol{\Pi}^j)$. Above, we denote $\alpha = d/(m\epsilon^2)$, $\alpha_j = d/(\mathrm{tr}(\boldsymbol{\Pi}^j)\epsilon^2)$, $\gamma_j = \mathrm{tr}(\boldsymbol{\Pi}^j)/m$, and $\epsilon > 0$ is a prescribed quantization error.

This coding rate measure utilizes local ϵ-ball packing to estimate the coding rate of the latent distribution from finite samples [13]. $R(\boldsymbol{Z})$, known as the *expansion* term, represents the total coding length of all features $\boldsymbol{Z}$ while $R_c(\boldsymbol{Z},\boldsymbol{\Pi})$, named *compression* term, measures the sum of coding lengths of each latent class distribution. They are called expansion and compression terms respectively, since to maximize ΔR, the first coding rate term is maximized and the second coding rate term is minimized.

The features learned by MCR^2 have precise statistical and geometric interpretations formalized in the following theorem.

Theorem 1.1.1. *(Theorem 2.1 in [25]) Suppose $\boldsymbol{Z}^* = \boldsymbol{Z}_1^* \cup \cdots \cup \boldsymbol{Z}_k^*$ is the optimal solution that maximizes the rate reduction (1.1) with the extra constraint that $\mathsf{rank}(\boldsymbol{Z}_j) \le d_j$, $\forall j \in \{1,\ldots,k\}$. As long as the ambient space is adequately large ($d \ge \sum_{j=1}^k d_j$), we have:*

1. *Between-class Discriminative: The subspaces are all orthogonal to each other, i.e. $(\boldsymbol{Z}_i^*)^\top \boldsymbol{Z}_j^* = 0$ for $i \neq j$.*

2. *Maximally Diverse Representation: As long as the coding precision is adequately high,* i.e., $\varepsilon^4 < \min_j \left\{ \frac{m_j}{m} \frac{d^2}{d_j^2} \right\}$, *each subspace achieves its maximal dimension,* i.e. $\mathsf{rank}(\boldsymbol{Z}_j^*) = d_j$. *In addition, the largest* $d_j - 1$ *singular values of* $\boldsymbol{Z}_j^*$ *are equal.*

In [25], it is demonstrated empirically that maximizing $\Delta R(\boldsymbol{Z})$ enforces each latent class distribution to be a low-dimensional subspace-like distribution of approximately $\frac{d}{k}$ dimension with class balance *i.e.* among all such discriminative representations, it prefers the one that spans the whole ambient space. The data points are distributed *isotropically* in each subspace except for possibly one dimension. In addition, these class distributions are orthogonal to each other (See Figure 1.1). Thus, by maximizing the coding rate difference, the features become *between-class* discriminative, whilst maintaining *intra-class* diversity. We refer interested readers to [25] for detailed proofs and empirical results.

1.2 Rate Reduction Network (ReduNet)

While an existing neural network architecture (such as ResNet) can be used for feature learning with MCR^2, a follow-up work [2] showed that a novel architecture can be explicitly constructed without back-propagation via emulating the projected gradient ascent scheme for maximizing $\Delta R(\boldsymbol{Z})$. This produces a "white box" network, called ReduNet, which has precise statistical and geometric interpretations. We review the construction of ReduNet as follows.

Let $\boldsymbol{Z}$ be initialized as the training data, i.e., $\boldsymbol{Z}_0 = \boldsymbol{X}$. Then, the projected gradient ascent step for optimizing the rate reduction $\Delta R(\boldsymbol{Z})$ in (1.1) is given by

$$\begin{aligned} \boldsymbol{Z}_{\ell+1} &\propto \boldsymbol{Z}_\ell + \eta \left(\frac{\partial \Delta R}{\partial \boldsymbol{Z}} \Big|_{\boldsymbol{Z}_\ell} \right) \\ &= \boldsymbol{Z}_\ell + \eta \Big(\boldsymbol{E}_\ell \boldsymbol{Z}_\ell - \sum_{j=1}^{k} \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{Z}_\ell^j \Big) \\ \text{s.t.} \quad & \|\boldsymbol{Z}_{\ell+1}^j\|_F^2 = \mathrm{tr}(\boldsymbol{\Pi}^j) = m_j \quad \forall j \in \{1, .., k\}, \end{aligned} \tag{1.4}$$

where we use $\boldsymbol{Z}_\ell^j = \boldsymbol{Z}_\ell \boldsymbol{\Pi}^j \in \mathbb{R}^{d \times m}$ to denote the feature matrix associated with the j-th class at the ℓ-th iteration, and $\eta > 0$ is the learning rate. The matrices $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ are obtained by evaluating the derivative $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ at $\boldsymbol{Z}_\ell$, given by

$$\boldsymbol{E}_\ell = \alpha \left(\boldsymbol{I} + \alpha \boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^\top \right)^{-1}, \tag{1.5}$$

$$\boldsymbol{C}_\ell^j = \alpha_j \left(\boldsymbol{I} + \alpha_j \boldsymbol{Z}_\ell^j \boldsymbol{Z}_\ell^{j\top} \right)^{-1}. \tag{1.6}$$

Observe that $\boldsymbol{E}_\ell \in \mathbb{R}^{d \times d}$ is applied to all features $\boldsymbol{Z}_\ell$ and it expands the coding length of the entire data. Meanwhile, $\boldsymbol{C}_\ell^j \in \mathbb{R}^{d \times d}$ is applied to features from class j, i.e., $\boldsymbol{Z}_\ell^j$, and it compresses the coding lengths of the j-th class.

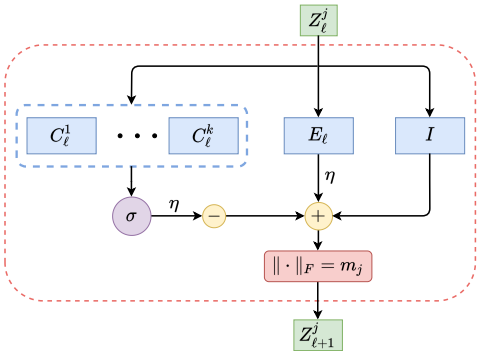


Figure 1.2: ReduNet Architecture in which we here adopt a slightly different normalization than [2] but is more suitable for the incremental learning as we will see in our derivation.

Each projected gradient step can be interpreted as one layer of a neural network composed of matrix multiplication and subtraction operators, with $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ being parameters associated with the ℓ-th layer computed using training data (See Figure 1.2). Then, given a test sample $\boldsymbol{x} \in \mathbb{R}^d$, its feature can be computed by setting $\boldsymbol{z}_0 = \boldsymbol{x}$ and iteratively carrying out the following incremental transform

$$\boldsymbol{z}_{\ell+1} \propto \boldsymbol{z}_\ell + \eta\Big(\boldsymbol{E}_\ell \boldsymbol{z}_\ell - \sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \boldsymbol{\pi}^j(\boldsymbol{z}_\ell)\Big). \tag{1.7}$$

Notice that the increment depends on $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)$, the membership of the feature $\boldsymbol{z}_\ell$, which is unknown for the test data. Therefore, [2] presented a method that replaces $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)$ in (1.7) by the following estimated membership

$$\hat{\boldsymbol{\pi}}_\ell^j(\boldsymbol{z}) = \frac{\exp\left(-\lambda k \|\boldsymbol{C}_\ell^j \boldsymbol{z}\|\right)}{\sum_{j=1}^k \exp\left(-\lambda k \|\boldsymbol{C}_\ell^j \boldsymbol{z}\|\right)} \in [0,1], \tag{1.8}$$

where $\lambda > 0$ is a confidence parameter. This leads to a nonlinear operator $\sigma\left(\boldsymbol{C}_\ell^1 \boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k \boldsymbol{z}_\ell\right) \doteq \sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \hat{\boldsymbol{\pi}}_\ell^j$ that, after being plugged into (1.7), produces a nonlinear layer as summarized in Figure 1.2. Stacking multiple such layers produces a multi-layer neural network for extracting discriminative features. Then, a nearest subspace classifier as we will discuss in Section 2.4 can classify the data. Note that each layer is interpretable and computed explicitly.

Note that when membership is known, ReduNet is simply a linear transformation for each class data. Additionally, note that $\boldsymbol{E}_l - \sum_{j=1}^k \gamma_j \boldsymbol{C}_l^i$ cannot move the data $\boldsymbol{Z}_l$ towards a direction that is not spanned by $\boldsymbol{Z}_l$. Thus, the final representation achieved by each class does not necessarily span all $\frac{d}{k}$ dimensions like the globally optimal configuration (Theorem 1.1.1) if the initial data lies on a lower dimensional subspace.

Chapter 2

ReduNet for Interpretable Incremental Learning

The work in this chapter was published at the Conference on Computer Vision and Pattern Recognition (CVPR) 2021 by the author, Ziyang Wu, Chong You, and Yi Ma [23].

2.1 Introduction

Humans are capable of acquiring new information continuously while retaining previously obtained knowledge. This seemingly natural capability, however, is extremely difficult for deep neural networks (DNNs) to achieve. *Incremental learning* (IL), also known as continual learning or life-long learning, thus studies the design of machine learning systems that can assimilate new information without forgetting past knowledge.

In incremental learning, models go through rounds of training sessions to accumulate knowledge for a particular objective (*e.g.* classification). Specifically, under *class* incremental learning (class-IL), an agent has access to training data from a subset of the classes, known as a *task*, at each training session and is evaluated on all seen classes at inference time. The overarching goal is to precisely fine-tune a model trained on previously seen tasks to additionally classify new classes of data. However, due to the absence of old data, such models often suffer from *catastrophic forgetting* [14], which refers to a drastic drop in performance after training incrementally on different tasks.

In the last few years, a flurry of continual learning algorithms have been proposed for DNNs, aiming to alleviate the effect of catastrophic forgetting. These methods can be roughly partitioned into three categories: 1) regularization-based methods that often involve knowledge distillation [12, 6, 19, 26], 2) exemplar-based methods that keep partial copies of data from previously learned tasks [16, 1, 22], and 3) modified architectures that attempt to utilize network components specialized for different tasks [17, 19, 11]. In practice, these algorithms exhibit varying performance across different datasets and their ability to mitigate catastrophic forgetting is inadequate. Factors including domain shift [18] across tasks and

imbalance of new and past classes [22] are part of the reason.

The fundamental roadblock in deep continual learning is that DNNs are trained and optimized in a "black box" fashion. Each model contains millions of mathematical operations and its complexity prevents humans from following the mapping from data input to prediction. Given our current limited understanding of network parameters, it is difficult, if not impossible, to precisely control the parameters of a pre-trained model such that the decision boundary learned fits to new data without losing its understanding of old data.

In this work, we take a drastically different approach to incremental learning. We avoid "black box" architectures entirely, and instead utilize a recently proposed "white box" DNN architecture derived from the principle of rate reduction [2]. Termed ReduNet, each layer of this DNN can be explicitly computed in a forward-propagation fashion and each parameter has precise statistical interpretations. The so-constructed network is intrinsically suitable for incremental learning because the second-order statistics of any previously-seen training data is preserved in the network parameters to be leveraged for future tasks.

We propose a new incremental learning algorithm utilizing ReduNet to demonstrate the power and scalability of designing more interpretable networks for continual learning. Specifically, we prove that a ReduNet trained incrementally can be constructed to be equivalent to one obtained by joint training, where all data, both new and old, is assumed to be available at training time. Finally, we observe that ReduNet performs significantly better on MNIST [9] and CIFAR-10 [7] in comparison to current continual DNN approaches.

2.2 Related Work

Since the early success of deep learning in classification tasks such as object recognition, attention has lately shifted to the problem of incremental learning in hopes of designing deep learning systems that are capable of continuously adapting to data from non-stationary and changing distributions.

Incremental learning can refer to different problem settings and most studies focus on three widely accepted scenarios [20]. Most of the earlier works [12, 17, 6, 19] study the *task* incremental (task-IL) setting, where a model, after trained on multiple tasks, must be able to classify on data belonging to all the classes it has seen so far. However, the model is additionally provided a task-ID indicating the task or subset of classes each datapoint belongs to. Models trained under this setting are thus required to distinguish among typically only a small number of classes. Recent works [24, 27] explore the more difficult *class* incremental (class-IL) setting, where task-ID is withheld at inference time. This setting is considerably more difficult since without the task-ID, each datapoint could potentially belong to any of the classes the model has seen so far. The other setting, known as *domain* incremental learning (domain-IL) differs from the previous two settings in that each task consists of all the classes the model needs to learn. Instead, a task-dependent transformation is applied to the data. For example, each task could contain the same training data rotated by differing

degrees and the model must learn to classify images of all possible rotations without access to the task-ID.

Deep continual learning literature from the last few years can be roughly partitioned into three categories as follows:

Regularization-based methods usually attempt to preserve some part of the network parameters deemed important for previously learned tasks. Knowledge distillation [4] is a popular technique utilized to preserve knowledge obtained in the past. Learning without Forgetting (LwF) [12], for example, attempts to prevent the model parameters from large drifts during the training of the current task by employing cross-entropy loss regularized by a distillation loss. Alternatively, elastic weight consolidation (EWC) [6] attempts to curtail learning on weights based on their importance to previously seen tasks. This is done by imposing a quadratic penalty term that encourages weights to move along directions with low Fisher information. Schwarz *et al*. [19] later proposed an online variant (oEWC) that reduces the cost of estimating the Fisher information matrix. Similarly, Zenke *et al*. [26] limits the changes of important parameters in the network by using an easy-to-compute surrogate loss during training.

Exemplar-based methods typically use a memory buffer to store a small set of data from previous tasks in order to alleviate catastrophic forgetting. The data stored is used along with the data from the current task to jointly train the model. Rebuffi *et al*. [16] proposed iCaRL which uses a herding algorithm to decide which samples from each class to store during each training session. This technique is combined with regularization with a distillation loss to further encourage knowledge retention [16]. A recent work by Wu *et al*. [22] achieved further improvements by correcting the bias towards new classes due to data imbalance, which they empirically show causes degradation in performance for large-scale incremental learning settings. This is accomplished by appending a bias-correction layer at the end of the network. Another increasingly popular approach is to train a generative adversarial network (GAN) [5, 21] on previously seen classes and use the generated synthetic data to facilitate training on future tasks.

Architecture-based methods either involve designing specific components in the architecture to retain knowledge of previously seen data or appending new parameters or entire networks when encountering new classes of data. Progressive Neural Network (PNN) [17], for example, instantiates a new network for each task with lateral connection between networks in order to overcome forgetting. This results in the number of networks to grow linearly with respect to the number of tasks as training progresses. Progress & Compress (P & C) [19] utilizes one network component to learn the new task, then distills knowledge into a main component that aggregates knowledge from previously encountered data. Li *et al*. [11], proposes a neural architecture search method that utilizes a separate network to learn whether to reuse, adapt, or add certain building blocks of the main classification network for each task encountered.

Our work studies the more difficult class-IL scenario and does not involve regularization or storing any exemplars. Our method thus can be characterized as an architecture-based approach. However, our method differs with the aforementioned works in several important aspects. First, we use a "white box" architecture that is computed exactly in a feed-forward

manner. Moreover, the network, when trained under class-IL scenario, can be shown to perform equivalently to one obtained from joint training while most existing works [11, 19, 17] based on modified architectures target the less challenging task-IL setting. We discuss the differences in more detail later in Section 2.3, after we have introduced our method properly.

2.3 Incremental Learning with ReduNet

In this paper, we tackle the task of *class* incremental learning, formalized as follows. Suppose we have a stream of tasks $\boldsymbol{D}^1, \boldsymbol{D}^2, \ldots, \boldsymbol{D}^t, \ldots$, where each task $\boldsymbol{D}^t$ consists of data from k_t classes, i.e, $\boldsymbol{D}^t = \{\boldsymbol{X}^{(t-1)\cdot k_t+1}, \ldots, \boldsymbol{X}^{t\cdot k_t}\}$ where $\boldsymbol{X}^j$ is a set of points in class j. The classes in different tasks are assumed to be mutually exclusive. Furthermore, it is assumed that the tasks arrive in an online setting, meaning that at timestep t when data $\boldsymbol{D}^t$ arrives, the data associated with old tasks $\{\boldsymbol{D}_i, i < t\}$ becomes unavailable. Therefore, the objective is to design a learning system that can adapt the model from the old tasks so as to correctly classify on all tasks hitherto, i.e., $\boldsymbol{D}^1, \ldots, \boldsymbol{D}^t$. In addition, we assume that we are not given the information on the task a test data belongs to, making this problem significantly more challenging than task-IL.

In this section, we show that ReduNet can perfectly adapt to a new task without forgetting old tasks. Specifically, we present an algorithm to adapt the ReduNet constructed from data $\{\boldsymbol{D}_i, i < t\}$ by using only the data in $\boldsymbol{D}^t$, so that the updated ReduNet is *exactly the same* as the ReduNet constructed as if data from all tasks $\{\boldsymbol{D}_i, i \leq t\}$ were available.

Derivation of Incrementally-Trained ReduNet

Without loss of generality, we consider the simple case with two tasks t and t' where t is treated as the old task and t' is treated as the new task. Assume that t and t' contain m_t, $m_{t'}$ training samples and k_t, $k_{t'}$ distinct classes, respectively. We denote such training data by $\boldsymbol{Z}_{0,t} \in \mathbb{R}^{d\times m_t}$ (for task t) and $\boldsymbol{Z}_{0,t'} \in \mathbb{R}^{d\times m_{t'}}$ (for task t'), and assume that they have been normalized by Frobenius norm as described in (1.4). For ease of notation, we label the classes as $\{1, ..., k_t\}$ for task t and $\{k_t + 1, ..., k_t + k_{t'}\}$ for task t'.

Let $\boldsymbol{\Theta}_t$ be the ReduNet of depth L trained on task t as described in Section 1.2. Given the new task $\boldsymbol{Z}_{0,t'}$, our objective is to train a network $\boldsymbol{\Theta}_{t\to t'}$ that adapts $\boldsymbol{\Theta}_t$ to have good performance for both tasks t and t'. Next, we show that a network $\boldsymbol{\Theta}_{t\to t'}$ can be constructed from $\boldsymbol{\Theta}_t$ and $\boldsymbol{Z}_{0,t'}$ such that it is *equivalent* to $\boldsymbol{\Theta}$ obtained from training on $\boldsymbol{Z}_0 = [\boldsymbol{Z}_{0,t}|\boldsymbol{Z}_{0,t'}] \in \mathbb{R}^{d\times m}$ where $m = m_t + m_{t'}$.

To start, consider the initial expansion term $\boldsymbol{E}_0 \in \mathbb{R}^{d\times d}$ and compression terms $\boldsymbol{C}_0^j$ at layer 0 of the joint network $\boldsymbol{\Theta}$ given by

$$\begin{aligned}\boldsymbol{E}_0 &= \alpha\left(\boldsymbol{I} + \alpha\boldsymbol{Z}_0\boldsymbol{Z}_0^\top\right)^{-1} \\ &= \alpha\left(\boldsymbol{I} + \alpha\big(\boldsymbol{Z}_{0,t}\boldsymbol{Z}_{0,t}^\top + \boldsymbol{Z}_{0,t'}\boldsymbol{Z}_{0,t'}^\top\big)\right)^{-1},\end{aligned} \tag{2.1}$$

and

$$\boldsymbol{C}_0^j = \begin{cases} \alpha_j \left(\boldsymbol{I} + \alpha_j \boldsymbol{Z}_{0,t}^j \boldsymbol{Z}_{0,t}^{j\top} \right)^{-1}, & \text{if } j \le k_t, \\ \alpha_j \left(\boldsymbol{I} + \alpha_j \boldsymbol{Z}_{0,t'}^j \boldsymbol{Z}_{0,t'}^{j\top} \right)^{-1}, & \text{else} \end{cases} \tag{2.2}$$

where $\alpha = d/(m\epsilon^2)$ and $\alpha_j = d/(\mathrm{tr}(\boldsymbol{\Pi}^j)\epsilon^2)$.

Note that the term $\boldsymbol{Z}_{0,t'}^j \boldsymbol{Z}_{0,t'}^{j\top}$ can be directly computed from input data $\boldsymbol{Z}_{0,t'}^j$. On the other hand, the term $\boldsymbol{Z}_{0,t}^j \boldsymbol{Z}_{0,t}^{j\top}$ cannot be directly computed from input data as $\boldsymbol{Z}_{0,t}^j$ is from the old task, which is no longer available under the IL setup. Our key observation is that $\boldsymbol{Z}_{0,t}^j \boldsymbol{Z}_{0,t}^{j\top}$ can be computed from the network $\boldsymbol{\Theta}_t$. Specifically, by denoting the compression matrices of $\boldsymbol{\Theta}_t$ as $\{\boldsymbol{C}_{\ell,t}^j\}$ for $\ell \in \{0, ..., L-1\}$, we have

$$\boldsymbol{Z}_{0,t}^j \boldsymbol{Z}_{0,t}^{j\top} = \left((\boldsymbol{C}_{0,t}^j / \alpha_j)^{-1} - \boldsymbol{I} \right) / \alpha_j. \tag{2.3}$$

Next, we show by induction that one can recursively compute $\boldsymbol{E}_\ell$ and $\{\boldsymbol{C}_\ell^j\}$ of $\boldsymbol{\Theta}$ for $\ell > 0$ from (2.1) and (2.2). To construct layer 1 of $\boldsymbol{\Theta}$, we observe that the output features of class j at layer 0 *before* normalization is as follows.

$$\begin{aligned} \boldsymbol{P}_0^j &= (\boldsymbol{Z}_0 + \eta \boldsymbol{E}_0 \boldsymbol{Z}_0 - \eta \sum_{i=1}^{k} \gamma_i \boldsymbol{C}_0^i \boldsymbol{Z}_0^i) \boldsymbol{\Pi}^j && (2.4) \\ &= \boldsymbol{Z}_0^j + \eta \boldsymbol{E}_0 \boldsymbol{Z}_0^j - \eta \gamma_j \boldsymbol{C}_0^j \boldsymbol{Z}_0^j && (2.5) \\ &= \big(\underbrace{\boldsymbol{I} + \eta \boldsymbol{E}_0 - \eta \gamma_j \boldsymbol{C}_0^j}_{\boldsymbol{L}_0^j \in \mathbb{R}^{d \times d}} \big) \boldsymbol{Z}_0^j. && (2.6) \end{aligned}$$

Notice the term $\boldsymbol{L}_0^j$ only depends on quantities already obtained at layer 0. To compute $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$, we need the covariance matrix of $\boldsymbol{P}_0^j$, which we observe to be

$$\boldsymbol{T}_1^j = \boldsymbol{P}_0^j \boldsymbol{P}_0^{j\top} = \boldsymbol{L}_0^j \boldsymbol{Z}_0^j \boldsymbol{Z}_0^{j\top} \boldsymbol{L}_0^{j\top}. \tag{2.7}$$

Notice that $\boldsymbol{T}_1^j$ can be expressed with known quantities of $\boldsymbol{L}_0^j$ and $\boldsymbol{Z}_{0,t}^j \boldsymbol{Z}_{0,t}^{j\top}$ if $j \le k_t$ or $\boldsymbol{Z}_{0,t'}^j \boldsymbol{Z}_{0,t'}^{j\top}$ if $j > k_t$. The remaining step would be to re-scale $\boldsymbol{T}_1^j$ as the updated representation $\boldsymbol{P}_0^j$ needs to be normalized to get $\boldsymbol{Z}_1^j$. Recall that we adopt the normalization scheme that imposes the Frobenius norm of each class $\boldsymbol{Z}_j$ to scale with m_j:

$$\|\boldsymbol{Z}_1^j\|_F^2 = m_j \iff \mathrm{tr}\big(\boldsymbol{Z}_1^j \boldsymbol{Z}_1^{j\top}\big) = m_j. \tag{2.8}$$

The re-scaling factor is then easy to calculate:

$$\boldsymbol{Z}_1^j \boldsymbol{Z}_1^{j\top} = \frac{m_j}{\mathrm{tr}(\boldsymbol{T}_1^j)} \boldsymbol{T}_1^j. \tag{2.9}$$

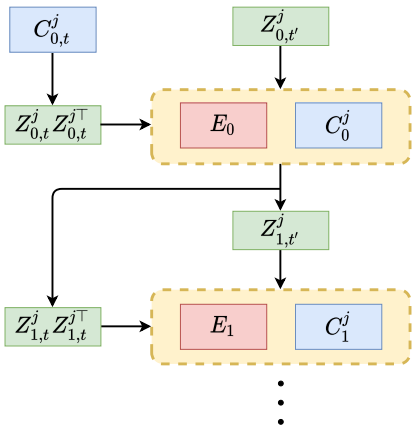


Figure 2.1: The joint network can be derived using simply $\boldsymbol{Z}_{0,t}\boldsymbol{Z}_{0,t}^\top$. We do not need the task t data $\boldsymbol{Z}_{0,t}$ directly.

From above, we see that we can obtain the correct value of the covariance matrix $\boldsymbol{Z}_1^j\boldsymbol{Z}_1^{j\top}$, from which we can derive $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$ for layer 1 of the joint network $\boldsymbol{\Theta}$ and obtain $\boldsymbol{Z}_{2,t'}^j$. With these values, we can compute $\boldsymbol{T}_2^j$.

By the same logic, we can recursively update $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ for all $\ell > 1$. Specifically, once we have obtained $\boldsymbol{Z}_{\ell-1}^j\boldsymbol{Z}_{\ell-1}^{j\top}$ and $\boldsymbol{L}_{\ell-1}^j$, it is straightforward to compute $\boldsymbol{T}_\ell^j = \boldsymbol{L}_{\ell-1}\boldsymbol{Z}_{\ell-1}^j\boldsymbol{Z}_{\ell-1}^{j\top}\boldsymbol{L}_{\ell-1}^{j\top}$ and therefore obtain

$$\boldsymbol{Z}_\ell^j\boldsymbol{Z}_\ell^{j\top} = \frac{m_j}{\operatorname{tr}(\boldsymbol{T}_\ell^j)}\boldsymbol{T}_\ell^j. \tag{2.10}$$

Note that we never need to access $\boldsymbol{Z}_{0,t} \in \mathbb{R}^{d\times m_t}$ directly. Instead, we iteratively update the covariance matrix $\boldsymbol{Z}_{\ell-1,t}^j\boldsymbol{Z}_{\ell-1,t}^{j\top} \in \mathbb{R}^{d\times d}$ for each class j using the procedure described. This concludes our induction and Algorithm 1 describes the entire training process for incremental learning on two tasks. The procedure is illustrated in Figure 2.1. This procedure can be naturally extended to settings with more than two tasks.

Comparison to Existing Methods

Incremental learning with ReduNet offers several nice properties: 1) Each parameter of the network has an explicit purpose, computed precisely to emulate the gradient ascent on the feature representation. 2) It does not require a memory buffer which is often needed in many state-of-the-art methods [16, 22, 1]. 3) It can be proven to behave like a network reconstructed from joint training, thus eliminating the problem of catastrophic forgetting.

Note that many existing works without relying on exemplars [12, 6, 19, 26] regularize the original weights of the model at each training session, effectively freezing certain parts of the

Algorithm 1 Incremental Learning with ReduNet

Input: Network $\boldsymbol{\Theta}_t$ with parameters $\boldsymbol{E}_{\ell,t}$ and $\{\boldsymbol{C}^j_{\ell,t}\}$, data $\boldsymbol{Z}^j_{0,t'}\ \forall j \in \{k_t+1, ..., k_t+k_{t'}\}$.
Compute $\boldsymbol{\Sigma}^j_{0,t} = \boldsymbol{Z}^j_{0,t}\boldsymbol{Z}^{j\top}_{0,t},\ \forall j \in \{1, ..., k_t\}$ by (2.3).
for $\ell = 0, 1, 2, ..., L-1$ **do**
  $\boldsymbol{\Sigma}_{\ell,t} = \sum_{j=1}^{k_t} \boldsymbol{\Sigma}^j_{\ell,t}$,
  $\boldsymbol{\Sigma}_{\ell,t'} = \sum_{j=k_t+1}^{k_t+k_{t'}} \boldsymbol{Z}^j_{\ell,t'}\boldsymbol{Z}^{j\top}_{\ell,t'}$,
  $\boldsymbol{E}_\ell = \alpha\big(\boldsymbol{I} + \alpha(\boldsymbol{\Sigma}_{\ell,t} + \boldsymbol{\Sigma}_{\ell,t'})\big)^{-1}$,
  $\boldsymbol{L}^j_\ell = \boldsymbol{I} + \eta\boldsymbol{E}_\ell - \eta\gamma_j\boldsymbol{C}^j_\ell \quad \forall j \in \{1, ..., k_t\}$,
  for $j = 1, 2, ..., k_t$ **do**
    $\boldsymbol{C}^j_\ell = \alpha_j\big(\boldsymbol{I} + \alpha_j\boldsymbol{\Sigma}^j_\ell\big)^{-1}$,
    $\boldsymbol{T}^j_{\ell+1,t} = \boldsymbol{L}^j_\ell\boldsymbol{\Sigma}^j_{\ell,t}\boldsymbol{L}^{j\top}_\ell$,
    $\boldsymbol{\Sigma}^j_{\ell+1,t} = \frac{m_j}{\mathrm{tr}\big(\boldsymbol{T}^j_{\ell+1,t}\big)}\boldsymbol{T}^j_{\ell+1,t}$.
  end for
  $\boldsymbol{Z}_{\ell+1,t'} \propto \boldsymbol{Z}_{\ell,t'} + \eta\boldsymbol{E}_\ell\boldsymbol{Z}_{\ell,t'} - \eta\sum_{i=k_t+1}^{k_t+k_{t'}} \gamma_i\boldsymbol{C}^i_\ell\boldsymbol{Z}^i_{\ell,t'}$
    s.t. $\|\boldsymbol{Z}^j_{\ell+1,t'}\|^2_F = m_j$.
end for
Output: Network $\boldsymbol{\Theta}$ with parameters $\boldsymbol{E}_\ell$ and $\{\boldsymbol{C}^j_\ell\}$.

network. Different tasks, however, tend to depend on different parts of the network, which eventually leads to conflicts on which parameters to regularize as the number of tasks to learn increases. These methods, as we see later in Figure 2.2, empirically perform sub-optimally in the class-IL setting. This in fact reveals the fundamental limitation that underlies in many incremental learning methods: a lack of understanding of how individual weights impact the learned representation of data points. ReduNet, on the other hand, sidesteps this problem by utilizing a fully interpretable architecture.

One notable property of ReduNet, at its current form, is that its width grows linearly with the number of classes as a new compression term $\boldsymbol{C}^j_\ell$ is appended to each layer whenever we see a new class. On the surface, this makes ReduNet similar to some architecture-based methods [11, 17] that dynamically expand the capacity of the network. However, there exists a major difference. ReduNet is naturally suited for class-IL scenario, whilst the aforementioned works do not address class-IL directly. Instead they only directly address task-IL, which they accomplish by optimizing a sub-network *per task*. These networks, which are designed to accomplish each task individually, fail to properly share information between the sub-networks to discriminate between classes of different tasks. ReduNet accomplishes class-IL by not only appending the class compression terms $\boldsymbol{C}^j$ to the network, but also modifying the expansion term $\boldsymbol{E}_\ell$ to share information about classes of all previous tasks.

For class-IL, such methods that also append new parameters to the architecture fail to completely address the problem of catastrophic forgetting. One can see why with a simple

example. Consider an ensemble learning technique where for each class j, we train an all-versus-one model that predicts whether a data point belongs to class j or not. At each task, we can feed the available data points into each model, labeled as 1 if it belongs in that class or 0 otherwise. However, by optimizing such "black box" models by back-propagation, we again arrive at the problem of catastrophic forgetting. Specifically, the model only sees training points of its own class only for one task or training session. For the remaining tasks, all data points that it must train will be of label 0, which prevents standard gradient descent from correctly learning the desired all-versus-one decision boundary, and there is no clear way to precisely address this optimization problem.

Although it is natural to expect the network to expand as the number of classes increases, it remains interesting to see if the growth of certain variations of the ReduNet can be sublinear instead of linear in the number of classes.

2.4 Experiments

We evaluate the proposed method on MNIST and CIFAR-10 datasets in a class-IL scenario and compare the results with existing methods. In short, for both MNIST and CIFAR-10, the 10 classes are split into 5 incremental batches or tasks of 2 classes each. After training on each task, we evaluate the model's performance on test data from all classes the model has seen so far. The same setting is applied to all other methods we compared to.

Datasets

We compare the incremental learning performance of ReduNet on the following two standard datasets.

MNIST [**10**]. MNIST contains 70,000 greyscale images of handwritten digits 0-9, where each image is of size 28×28. The dataset is split into training and testing sets, where the training set contains 60,000 images and the testing dataset contains 10,000 images.

CIFAR-10 [**8**]. CIFAR-10 contains 60,000 RGB images of 10 object classes, where each image is of size 32×32. Each class has 5,000 training images and 1,000 testing images. We normalize the input data by dividing the pixel values by 255, and subtracting the mean image of the training set.

Implementation Details

We implement ReduNet for each training dataset in the following manner.

ReduNet on MNIST. To construct a ReduNet on MNIST, we first flatten the input image and represent it by a vector of dimension 784. Then, with a precision $\epsilon = 0.5$ in the MCR^2 objective (1.1), we apply 200 iterations of projected gradient iterations to compute $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ matrices for each iteration ℓ. The learning rate is set to $\eta = 0.5 \times 0.933^\ell$ at the ℓ-th iteration. These matrices are the parameters of the constructed ReduNet. Given a test

data, its feature can be extracted with the incremental transform in (1.7) with estimated labels computed as in (1.8) with parameter $\lambda = 1$. At each training session, we update the ReduNet by the procedure described in Algorithm 1.

We note that hyper-parameter tuning in ReduNet does not require a training/validation splitting as in regular supervised learning methods. The hyper-parameters described above for ReduNet are chosen based on the training data. This is achieved by evaluating the estimated label through (1.8) on the training data, and comparing such labels with ground truth labels. Then, the model parameter ϵ, learning rate η and the softmax confidence parameter λ are chosen as those that gives the highest accuracy with the estimated labels (at the final layer).

ReduNet on CIFAR-10. We apply 5 random Gaussian kernels with stride 1, size 3×3 on the input RGB images.[1] This lifts each image to a multi-channel signal of size $32 \times 32 \times 5$, which is subsequently flattened to be a $\mathbb{R}^{5,120}$ dimensional vector. Subsequently, we construct a 50-layer ReduNet with all other hyper-parameters the same as those for MNIST. All hyperparameters stated above, including the depth of the network, were chosen such that the ΔR loss has sufficiently converged.

Comparing Methods. We compare our approach to the following state of the art algorithms: iCaRL [16], LwF [12], oEWC [19], SI [26] and DER [1]. For these algorithms, we utilize the same benchmark and training protocol as Buzzega *et al*. [1]. For MNIST, we employ a fully-connected network with two hidden layers comprised of 100 ReLU units. For CIFAR-10, we rely on ResNet18 without pre-training [3]. All the networks were trained by stochastic gradient descent. For MNIST, we train on one epoch per task. For CIFAR-10, we train on 100 epochs per task. The number of epochs were chosen based on the complexity of the dataset. For each algorithm, batch size, learning rate, and specific hyperparameters for each algorithm were selected by performing a grid-search using 10% of the training data as a validation set and selecting the hyperparameter that achieves the highest final accuracy. The optimal hyperparameters utilized for the benchmark experiments are reported in [1].

The performance of state of the art algorithms utilizing a replay buffer highly depends on the number of exemplars, or samples from previous tasks, it is allowed to retain. We test on two exemplar-based algorithm, iCaRL and DER. For both MNIST and CIFAR-10, we set the total number of exemplars to 200.

Nearest Subspace Classification

By the principle of maximal rate reduction, the ReduNet $f(\boldsymbol{X}, \theta)$ extracts features such that each class lies in a low-dimensional linear subspace and different subspaces are orthogonal. As suggested by the original MCR^2 work [2], we utilize a nearest subspace classifier to classify the test data featurized to maximize ΔR. Given a test sample $\boldsymbol{z}_{test} = f(\boldsymbol{x}_{test}, \theta)$, the label

[1] This choice is limited by our current computational resources. Although this choice is not adequate to achieve top classification performance, it is adequate to verify the advantages of our method in the incremental setting.

Algorithm	MNIST					CIFAR-10				
	Task 1	Task 2	Task 3	Task 4	Task 5	Task 1	Task 2	Task 3	Task 4	Task 5
LwF	0.999	0.009	0.0	0.0	0.0	0.979	0.0	0.0	0.0	0.0
oEWC	**1.0**	0.004	0.0	0.0	0.0	0.981	0.0	0.0	0.0	0.0
SI	0.997	0.004	0.001	0.0	0.0	**0.989**	0.0	0.0	0.0	0.0
iCaRL (200 Exemplars)	0.999	0.806	0.708	0.612	0.596	0.964	0.720	0.427	0.362	0.313
DER (200 Exemplars)	0.999	0.967	0.941	0.883	0.735	0.985	**0.816**	0.608	0.404	0.292
ReduNet(Ours)	0.999	**0.994**	**0.993**	**0.989**	**0.987**	0.875	0.754	**0.714**	**0.642**	**0.562**
Upper Bound (UB)	0.999	0.995	0.990	0.988	0.982	0.989	0.971	0.957	0.963	0.920

Table 2.1: Test Accuracy (%) on Task 1 After Each Training Session on MNIST and CIFAR-10.

predicted by a nearest subspace classifier is

$$y = \underset{y\in 1,\ldots,k}{\arg\min} \left\| (\boldsymbol{I} - \boldsymbol{U}^y \boldsymbol{U}^{y\top}) \boldsymbol{z}_{test} \right\|_2^2, \tag{2.11}$$

where $\boldsymbol{U}^y$ is a matrix containing the top x principal components of the covariance of the training data passed through ReduNet, *i.e.* $\boldsymbol{Z}_{train}\boldsymbol{Z}_{train}^\top$ for $\boldsymbol{Z}_{train} = f(\boldsymbol{X}_{train}, \theta)$.

Since we do not have access to $\boldsymbol{Z}_{train}$ during evaluation, we instead collect the $\boldsymbol{C}^j$ matrices at the very last layer L and extract the covariance matrix $\boldsymbol{\Sigma}_L^j$ to be further processed by SVD. For MNIST, we utilize the top 28 principal components. For CIFAR-10, we utilize the top 15 principal components.

Results and Analysis

In this section, we evaluate the class-IL performance of incremental ReduNet against three regularization-based methods (oEWC, SI, LwF) and two replay-based methods leveraging 200 exemplars (iCaRL, DER) on MNIST and CIFAR-10. We also provide the upper bound (UB) achieved by joint training a model utilizing the same architecture as the baseline methods. After the model is trained on each task, performance is evaluated by computing the accuracy on test data from all classes the model has seen so far. To observe the degree of forgetting, we record the model's performance on Task 1 after training on each subsequent task. For both MNIST and CIFAR-10, we observe a substantial performance increase by utilizing incremental ReduNet as shown in Figure 2.2. Additionally, we observe ReduNet shows significantly less forgetting (see Table 2.1).

On MNIST, we observe a 3% decay in accuracy across the tasks on ReduNet versus a 20-80% decay on benchmark methods (Figure 2.2). We measure *decay* as the difference in average accuracy between the first and last task. ReduNet retains a classification accuracy of 96%. This is of no surprise since MNIST is relatively linearly separable, allowing second-order information about the data to be sufficient for ReduNet to correctly classify the digits. We observe that even for a very simple task as MNIST, competing continual learning

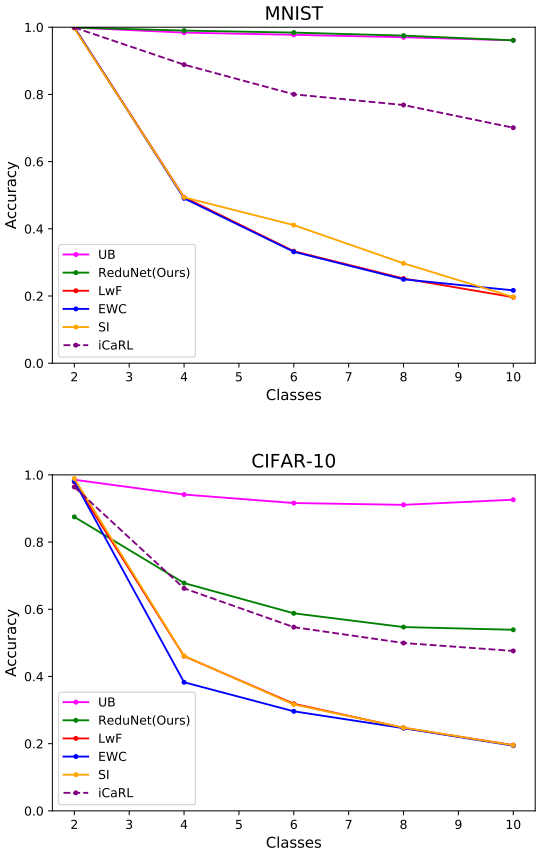


Figure 2.2: Incremental learning results (accuracy) on MNIST and CIFAR-10. Both datasets have 5 incremental batches. We also provide the upper bound (UB) given by joint training a model utilizing the same architecture as the baseline methods. In *solid* lines are regularization-based methods and in *dashed* are exemplar-based methods, which saves 200 samples from previous tasks. Note that the decay in the performance in ReduNet is simply because classification is harder to accomplish with more classes, not because of catastrophic forgetting.

algorithms fail spectacularly due to catastrophic forgetting. In fact, as shown in Table 2.1, models trained by regularization-based methods retain 0% accuracy for classes 0 and 1 after incrementally training on digits 2 to 5 (up to Task 3). The drastic decay in performance of benchmark methods is expected and replicated often in class-IL literature [1, 24]. Note that ReduNet observes no catastrophic forgetting and the decay in performance is due to the fact that classification is increasingly harder to accomplish with more classes.

Similar improvement in performance utilizing ReduNet is also observed on CIFAR-10, a more complex image dataset. We observe a 42-80% decay in accuracy for benchmark methods, whereas incremental ReduNet observes a 34% decrease (in Figure 2.2). The

algorithm that achieves the closest performance to ReduNet is iCaRL and DER, exemplar-based methods that require access to 200 previously observed exemplars. Certainly, as can be seen by the 88% accuracy on Task 1 of CIFAR-10, ReduNet at its current basic form (only using 5 randomly initialized kernels without back-propagation) is not able to reach the same classification accuracy as ResNet-18 for complex image classification tasks. It is thus not surprising that DER, based on more established network architectures, exceeds ReduNet in terms of average accuracy. However, as shown in Table 2.1, ReduNet decays gracefully and significantly outperforms other methods in terms of forgetting, retaining over 55% accuracy on Task 1 versus less than 30% by DER.

We note that ReduNet is currently a slower training framework given its current naive implementation using CuPy. Utilizing a single NVIDIA TITAN V GPU, each task training session took approximately 1500 seconds for MNIST and 9200 seconds for CIFAR10. On the other hand, joint training a model by back-propagation for each task took 23 and 2500 seconds for MNIST and CIFAR10, respectively.

2.5 Conclusions and Future Work

We demonstrated through an incremental version of the recently proposed ReduNet, the promise of leveraging interpretable network design for continual learning. The proposed network has shown significant performance increases in both synthetic and complex real data, even without utilizing any fine-tuning with back-propagation. It has clearly shown that if knowledge of past learned tasks are properly utilized, catastrophic forgetting needs not to happen as new tasks continue to be learned.

We purpose of this work is not to push the state of art classification accuracy or efficiency on any single large-scale real-world dataset. Rather we want to use the simplest experiments to show beyond doubt the remarkable effectiveness and great potential of this new framework. Using CIFAR-10 as an example, simply utilizing a relatively small set of 5 random lifting kernels was already sufficient for a decent incremental classification performance. We believe that to achieve better performance for more complex tasks and datasets, judicious design or learning of more convolution kernels would be needed. This leaves plenty of room for further improvements.

This work also opens up a few promising new extensions. As we have mentioned earlier, the current framework requires the width of the network to grow linearly in the number of classes. It would be interesting to see if some of the filters can be shared among old/new classes so that the growth can be sublinear. In the next chapter, we will discuss several approaches to decrease the memory footprint of incremental ReduNet.

To a large extent, the rate reduction gives a unified measure for learning discriminative representations in supervised, semi-supervised, and unsupervised settings. We believe our method can be easily extended to cases when some of the new data do not have class information.

Chapter 3

Low Rank Approximation for Efficient ReduNet

As we've observed in the previous chapter, incremental ReduNet tends to have a large memory footprint that increases linearly in the number of classes k. Namely, given d dimensional data, each layer requires one to save $k+1$ $d \times d$ weight matrices: $\boldsymbol{E}_{l,t}$, $\{\boldsymbol{C}_{l,t}^j\}$. In this chapter, we discuss naive, yet effective methods at reducing this memory footprint.

3.1 Low rank approximation

Though real world data such as images are often high dimensional, we often expect the data from each class to lie on low dimensional curved manifolds. For example, for MNIST and CIFAR-10, the singular values of the sample covariance matrix of each class is very small across most dimensions (Figure 3.1). Thus, the matrices are approximately low-rank.

We may utilize this understanding to find a substantially compressed version of ReduNet with equal performance. Namely, note that given that the distribution of class j, $\boldsymbol{Z}_j\boldsymbol{Z}_j^\top$, is low rank, $\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j$ is also low rank where $\boldsymbol{C}_j$ is the corresponding compression matrix.

$$\begin{aligned} \frac{1}{\alpha_j}\boldsymbol{C}_j = (\boldsymbol{I} + c_j\boldsymbol{Z}_j\boldsymbol{Z}_j^\top)^{-1} = \boldsymbol{U_j}\mathsf{diag}\left(\frac{1}{1+c_j\lambda_{ji}}\right)\boldsymbol{U_j}^\top \\ \Rightarrow \boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j = \boldsymbol{U_j}\mathsf{diag}\left(1 - \frac{1}{1+c_j\lambda_{ji}}\right)\boldsymbol{U_j}^\top \end{aligned} \tag{3.1}$$

We may find a low rank approximation of the compression matrix by optimizing over the following objective.

$$\begin{aligned} \min_{\boldsymbol{D}_j} \left\| (\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j) - \boldsymbol{D}_j \right\|_F^2 \\ \text{subject to}\, \mathsf{rank}(\boldsymbol{D}_j) < \frac{d}{k} \end{aligned} \tag{3.2}$$

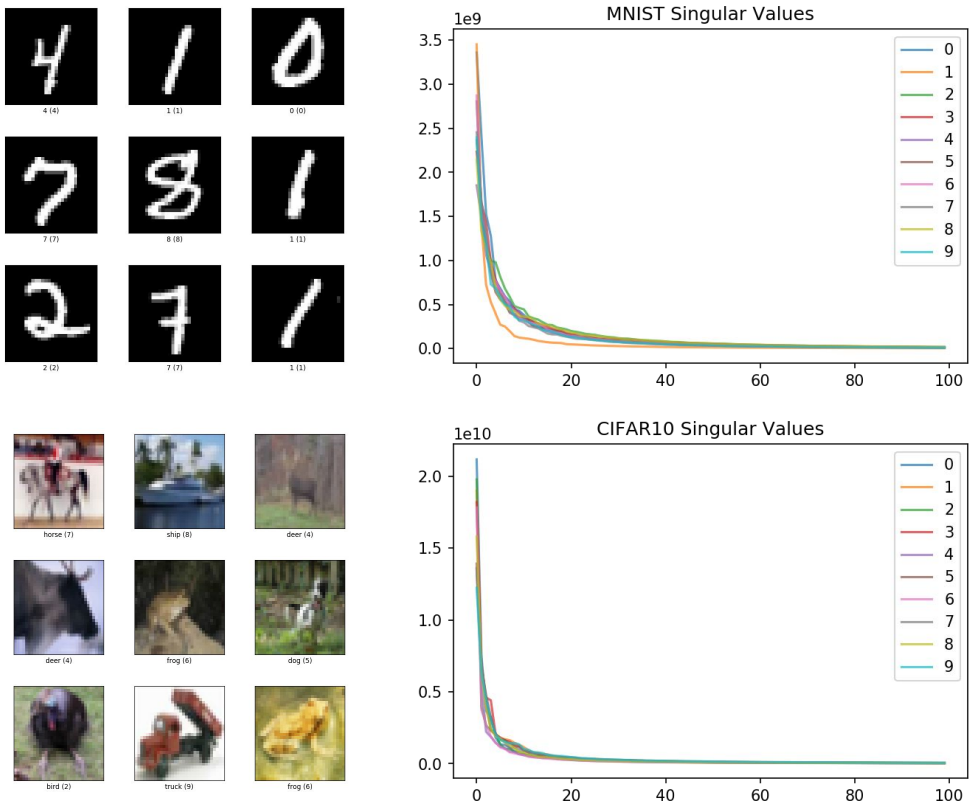


Figure 3.1: The first 100 eigenvalues of each class distribution of MNIST and CIFAR10. MNIST is $32 \times 32 = 784$ dimensional and CIFAR10 is $28 \times 28 \times 3 = 3072$ dimensional. Note that eigenvalues nearly decay to 0 by the 20th dimension for both datasets.

where d is the ambient dimension and k is the number of classes.The rank constraint is chosen based off the fact that with class balance, the globally optimal configuration $\boldsymbol{Z}_j^*$ spans at most a $\frac{d}{k}$ dimensional subspace.

The above objective has a closed form solution given by $\boldsymbol{D}_j = \boldsymbol{U}_j[:\frac{d}{k}]\boldsymbol{\lambda}_j[:\frac{d}{k}]\boldsymbol{U}_j[:\frac{d}{k}]^\top$ where $\boldsymbol{U}_j[:\frac{d}{k}]$ and $\boldsymbol{\lambda}_j[:\frac{d}{k}]$ are the top $\frac{d}{k}$ eigenvectors of $\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j$ and their corresponding eigenvalues, respectively.

To be more flexible with the rank constraint, we may alternatively approximate $\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j$ up to a certain tolerance

$$\frac{\left\|(\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j) - \boldsymbol{D}_j\right\|_F^2}{\left\|(\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j)\right\|_F^2} \leq \tau \tag{3.3}$$

and greedily choose the the eigenvectors of $\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j$ until the approximation is below the tolerance. Specifically,

Algorithm 2 Greedy Low Rank Approximation

Input: ε, $\boldsymbol{Z}_j$
$r = 1$, $error = \infty$
$\boldsymbol{U}, \boldsymbol{\lambda}, \boldsymbol{V} = \mathsf{SVD}(\boldsymbol{Z}_j \boldsymbol{Z}_j^T)$
while $error > \varepsilon$ **do**
    $\boldsymbol{D}_j = \boldsymbol{U}[:r]\mathsf{diag}(\boldsymbol{\lambda}[:r])\boldsymbol{V}[:r]^T$
    $error = \frac{\left\|(\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}_j)-\boldsymbol{D}_j\right\|_F^2}{\left\|(\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}_j)\right\|_F^2}$
    $r+ = 1$
end while
Output: $\boldsymbol{U}[:r], \boldsymbol{\lambda}[:r], \boldsymbol{V}[:r]$

From $\boldsymbol{D}_j$, we may approximately recover $\boldsymbol{C}_j$ by doing $\boldsymbol{C}_j = \alpha_j(\boldsymbol{I} - \boldsymbol{D}_j)$. By saving $r \leq \frac{d}{k}$ top eigenvalues and corresponding eigenvectors of $\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j$ the memory required to save each layer of ReduNet *at least* reduces to

$$\textbf{Original: } (k+1) \times d \times d \Longrightarrow \textbf{Compressed: } 2 \times d \times d + d$$

The extra d term comes from saving the eigenvalues. Note that the memory of this compressed network is now independent on the number of classes.

Experiments

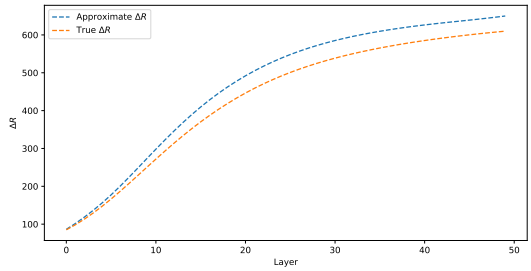
MNIST

We run 50-layer Redunet on MNIST with 1000 training samples from each of the 10 classes. We set learning rate to be $\eta = 0.1$ and ball packing error to be $\varepsilon^2 = 0.5$. Each image is $28 \times 28 = 784$, so the dimension of $\boldsymbol{E}$ and $\boldsymbol{C}_j$ is 784×784. We derive the compressed network by saving the $\frac{d}{k}$-rank approximation of the compression matrices $\{\boldsymbol{C}_l^j\}$ at each layer l.

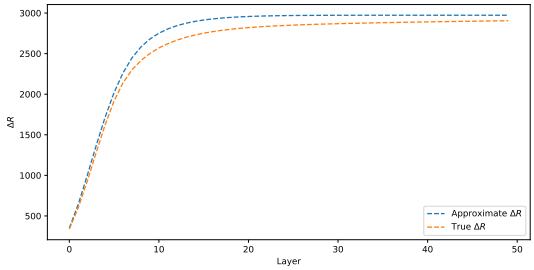
CIFAR10

For CIFAR10, we run a 50-layer Redunet with 1000 training samples from each of the 10 classes. We set learning rate to be $\eta = 0.1$ and ball packing error to be $\varepsilon^2 = 0.5$. Each image is $32 \times 32 \times 3 = 3072$ dimensions, so the dimension of $\boldsymbol{E}$ and $\boldsymbol{C}_j$ is 3072×3072. We derive the compressed network by saving the $\frac{d}{k}$-rank approximation of the compression matrices $\{\boldsymbol{C}_l^j\}$ at each layer l.

We compare the losses and feature representations between original ReduNet and approximate



(a) MNIST



(b) CIFAR10

Figure 3.2: We pass through the training data of MNIST and CIFAR10 through the original ReduNet and compressed ReduNet, and compare the MCR loss $\Delta R(\boldsymbol{Z}_l)$ at each layer l. We observe the loss curve for compressed ReduNet to be better than that of true ReduNet

ReduNet. Recall that the class distributions of MNIST are already quite low rank and separable (Figure 3.1). Since the class distributions of CIFAR10 and MNIST are low dimensional, we observe that the low rank approximations of the compression matrices $\boldsymbol{C}_l^j$ are close to the original matrices. Furthermore, compressed ReduNet achieved better loss curves than true ReduNet. By compressing ReduNet up to some tolerance, we were able to reduce the memory footprint of the network even further 3.3.

3.2 Block Diagonal Approximation by Eigenspace

We may further compress the model from our knowledge that the eigenvectors of each class distribution $\boldsymbol{Z}_j\boldsymbol{Z}_j^\top$ must be some linear combination of the eigenvectors of the joint distribution $\boldsymbol{Z}\boldsymbol{Z}^\top = \sum_j \boldsymbol{Z}_j\boldsymbol{Z}_j^\top$. In fact, at a critical point $\boldsymbol{Z}^*$ of the loss ΔR, the eigenvectors of $\boldsymbol{Z}\boldsymbol{Z}^\top$ is precisely equal to the eigenvectors of $\boldsymbol{Z}_j\boldsymbol{Z}_j^\top$. This statement is formalized in the following theorem.

Theorem 3.2.1. *If $\boldsymbol{Z}^*$ is a local maxima of $\Delta R(\boldsymbol{Z})$, then each eigenvector with positive*

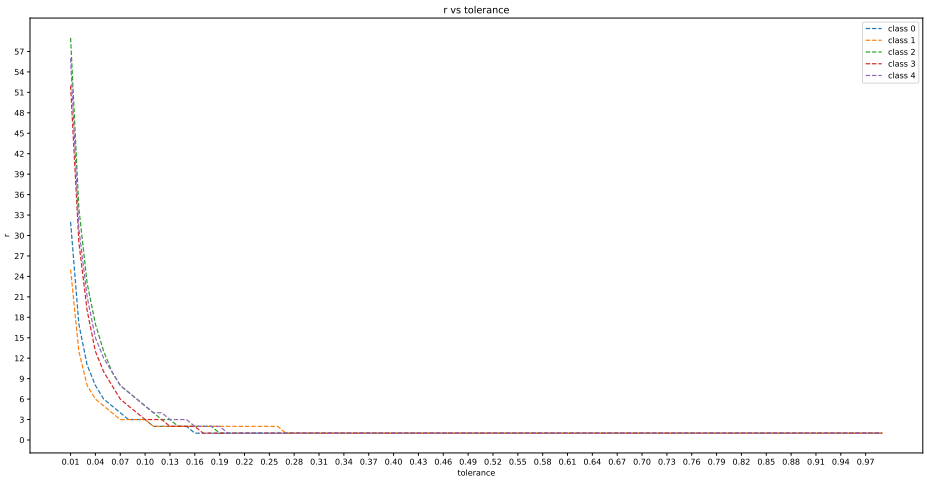


Figure 3.3: In MNIST, we observe that only a few eigenvectors are necessary to approximate C_j of the first layer for small tolerance values.

corresponding eigenvalue of the class subspace $\boldsymbol{Z}_j^* \boldsymbol{Z}_j^{*\top}$ *must be an eigenvector of the whole distribution* $\boldsymbol{Z}^* \boldsymbol{Z}^{*\top}$.

Proof. Recall MCR^2 objective:

$$\max \Delta R(\boldsymbol{Z}) = R(\boldsymbol{Z}, \epsilon) - R^c(\boldsymbol{Z}, \epsilon | \boldsymbol{\Pi}) \quad \text{s.t.} \quad \|\boldsymbol{Z}_j\|_F^2 = m_j \quad \forall j \in \{1, ..., k\} \tag{3.4}$$

By first order necessary conditions, note that $\boldsymbol{Z}$ is a stationary point if for each class distribution $\nabla_{\boldsymbol{Z}_j} \Delta R = a_j \boldsymbol{Z}_j$ for some constant $a_j \in \mathbb{R}$. Recall that the gradient is

$$\frac{\partial \Delta R}{\partial \boldsymbol{Z}_j} = \left(\boldsymbol{E} - \gamma_j \boldsymbol{C}_\ell^j \right) \boldsymbol{Z}_j \tag{3.5}$$

So at a stationary point, each sample $\boldsymbol{z}_i^j$ in $\boldsymbol{Z}_j$ for $i \in 1, ..., m_j$ satisfies

$$(\boldsymbol{E} - \gamma_j \boldsymbol{C}_j) \boldsymbol{z}_i^j = a_j \boldsymbol{z}_i^j.$$

Furthermore, note that each eigenvector $\boldsymbol{u}_l^j$ of $\boldsymbol{Z}_j \boldsymbol{Z}_j^\top$ for $0 \leq l \leq d$ with corresponding eigenvalue $\lambda_l^j > 0$, is some linear combination of class samples

$$\boldsymbol{u}_l^j = \sum_{i=1}^{m_j} \alpha_i \boldsymbol{z}_i^j.$$

Thus,

$$\begin{aligned}
&(\boldsymbol{E}-\gamma_j\boldsymbol{C}_j)\boldsymbol{u}_l^j \\
&=\sum_{i=1}^{m_j}\alpha_i(\boldsymbol{E}-\gamma_j\boldsymbol{C}_j)\boldsymbol{z}_i^j \\
&=a_j\sum_{i=1}^{m_j}\alpha_i\boldsymbol{z}_i^j \\
&=a_j\boldsymbol{u}_l^j
\end{aligned} \tag{3.6}$$

Since $\boldsymbol{C}_j\boldsymbol{u}_l^j=\alpha_j(1+c_j\lambda_l^j)^{-1}\boldsymbol{u}_l^j$ and $(\boldsymbol{E}-\gamma_j\boldsymbol{C}_j)\boldsymbol{u}_l^j=a_j\boldsymbol{u}_l^j$, we may conclude that

$$\begin{aligned}
\boldsymbol{E}\boldsymbol{u}_l^j&=a_j\boldsymbol{u}_l^j+\gamma_j\boldsymbol{C}_j\boldsymbol{u}_l^j \\
&=\left(a_j+\gamma_j\alpha_j(1+c_j\lambda_l^j)^{-1}\right)\boldsymbol{u}_l^j.
\end{aligned} \tag{3.7}$$

Therefore, the eigenvectors (with positive eigenvalue) of each class subspace $\boldsymbol{Z}_j\boldsymbol{Z}_j^\top$ must also be an eigenvector of the whole distribution $\boldsymbol{Z}\boldsymbol{Z}^\top$. □

Thus, we may simply utilize the eigenvectors of $\boldsymbol{I}-\frac{1}{\alpha}\boldsymbol{E}=\boldsymbol{U}\left(\boldsymbol{I}-(\boldsymbol{I}+\alpha\boldsymbol{\Sigma})^{-1}\right)\boldsymbol{U}^\top$ as a dictionary for the model, and use them to approximate the eigenvectors of $\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}=\boldsymbol{U}_j\left(\boldsymbol{I}-(\boldsymbol{I}+\alpha_j\boldsymbol{\Sigma}_j)^{-1}\right)\boldsymbol{U}_j^\top$. Additionally, at a critical point, we know that the eigenvectors of $\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}_j$ with positive corresponding eigenvalues is equal to approximately $\frac{d}{k}$ of the eigenvalues of $\boldsymbol{I}-\frac{1}{\alpha}$ Formally, we solve the following objective

$$\begin{aligned}
&\min_{\boldsymbol{D}_j}\left\|(\boldsymbol{I}-\boldsymbol{C}_j)-\boldsymbol{U}\boldsymbol{D}_j\boldsymbol{U}^\top\right\|_F^2 \\
&\text{subject to}\,\mathsf{rank}(\boldsymbol{D}_j)<\frac{d}{k}
\end{aligned} \tag{3.8}$$

Similar to low-rank approximation, we may be more flexible with the rank constraint by approximating $\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}_j$ up to a certain tolerance

$$\frac{\left\|(\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}_j)-\boldsymbol{U}\boldsymbol{D}_j\boldsymbol{U}^\top\right\|_F^2}{\left\|(\boldsymbol{I}-\frac{1}{\alpha_j}\boldsymbol{C}_j)\right\|_F^2}\leq\tau. \tag{3.9}$$

We may solve the objective up to some tolerance using the following greedy algorithm (Algorithm 3).

Essentially, the algorithm projects each eigenvector $u\in\boldsymbol{U}$ of $\boldsymbol{I}-\boldsymbol{E}$ onto each class subspace $\|(\boldsymbol{I}-\boldsymbol{C}_j)u\|_2$ and selects the top $r\leq\frac{d}{k}$ eigenvectors in $\boldsymbol{I}-\frac{1}{\alpha}\boldsymbol{E}$ with largest

Algorithm 3 Greedy Block Diagonal Approximation

Input: ε, $\boldsymbol{Z}_j$, $\boldsymbol{Z}$
$r = 1$, $error = \infty$
$\boldsymbol{U}, \boldsymbol{\lambda}, \boldsymbol{V} = \mathsf{SVD}(\boldsymbol{Z}\boldsymbol{Z}^T)$
Sort columns u_i of $\boldsymbol{U}$ by $\left\|(\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j)u_i\right\|_2^2$ in descending order
while $error > \varepsilon$ **do**
    $\boldsymbol{D}_j = \min_{\boldsymbol{D}_j} \left\|(\boldsymbol{I} - \boldsymbol{C}_j) - \boldsymbol{U}[:r]\boldsymbol{D}_j\boldsymbol{U}[:r]^\top\right\|_F^2 = \boldsymbol{U}[:r]^\top(\boldsymbol{I} - \boldsymbol{C}_j)\boldsymbol{U}[:r]$
    $error = \dfrac{\left\|(\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j) - \boldsymbol{U}\boldsymbol{D}_j\boldsymbol{U}^\top\right\|_F^2}{\left\|(\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j)\right\|_F^2}$
    $r \mathrel{+}= 1$
end while
$\boldsymbol{\pi}_j$ = Indices corresponding to columns of $\boldsymbol{U}$ chosen
Output: $\boldsymbol{D}_j$, $\boldsymbol{\pi}_j$

projection to approximate $\boldsymbol{I} - \frac{1}{\alpha_j}\boldsymbol{C}_j$. By saving $\boldsymbol{D}_j \in \mathbb{R}^{r\times r}$ and $\boldsymbol{\pi} \in \mathbb{R}^r$ for $r \leq \frac{d}{k}$ instead of $\boldsymbol{C}_j$, the memory *at least* reduces to

$$\textbf{Original: } (k+1)\times d\times d \Longrightarrow \textbf{Compressed: } k\times r\times r + d\times d + k\times d = (1+\frac{1}{k})\times d\times d + d$$

Additionally, we may reduce the computation needed during forward passing of test data at the label approximation layers. Specifically, once $\boldsymbol{D}_j$ is computed, we can save computation by no longer requiring passing each $\boldsymbol{C}_j$ with data $\boldsymbol{Z}$ every time. Concretely, given $\boldsymbol{A} = \boldsymbol{U}^\top\boldsymbol{Z}$, we can obtain $\boldsymbol{E}\boldsymbol{Z} = \boldsymbol{U}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^\top\boldsymbol{Z} = \boldsymbol{U}\boldsymbol{\Sigma}^{-1}\boldsymbol{A}$.

Also, we can compute $\boldsymbol{C}_j\boldsymbol{Z} = (\boldsymbol{I} - (\boldsymbol{I} - \boldsymbol{C}_j)_{approx})\,\boldsymbol{Z} = \boldsymbol{Z} - \boldsymbol{U}[:,\boldsymbol{\pi}_j]\boldsymbol{D}_j^\top\boldsymbol{A}[\boldsymbol{\pi}_j,:]$ where $\boldsymbol{U}[:,\boldsymbol{\pi}_j]$ is selecting columns of $\boldsymbol{U}$ and $\boldsymbol{A}[\boldsymbol{\pi}_j,:]$ is selecting rows of $\boldsymbol{A}$ given indices $\boldsymbol{\pi}_j$. The expensive operation of multiplying a $d\times d$ matrix with the data matrix of dimension $d\times m$ only needs to occur once for each ReduNet layer the data passes through.

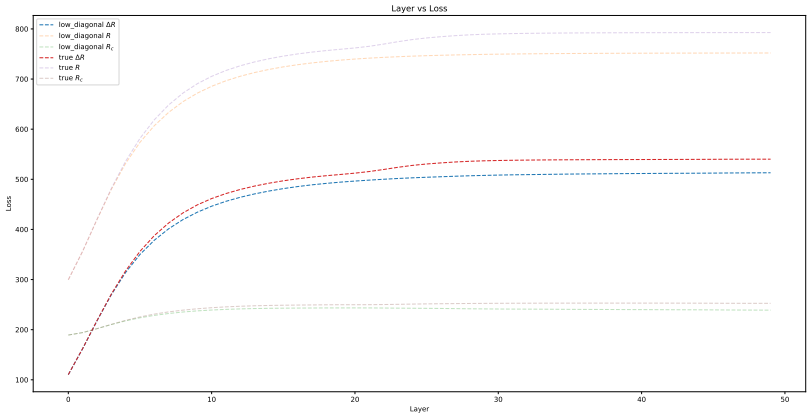
Namely, the computational complexity reduces to

$$\textbf{Original: } O((k+1)d^2n) \Longrightarrow \textbf{Compressed: } \underbrace{O(2d^2n)}_{\boldsymbol{A}+\boldsymbol{E}\boldsymbol{Z}} + \underbrace{O(kdrn) + O(kdr^2)}_{\boldsymbol{C}_j\boldsymbol{Z}} = O(3d^2n) + O(kdr^2)$$

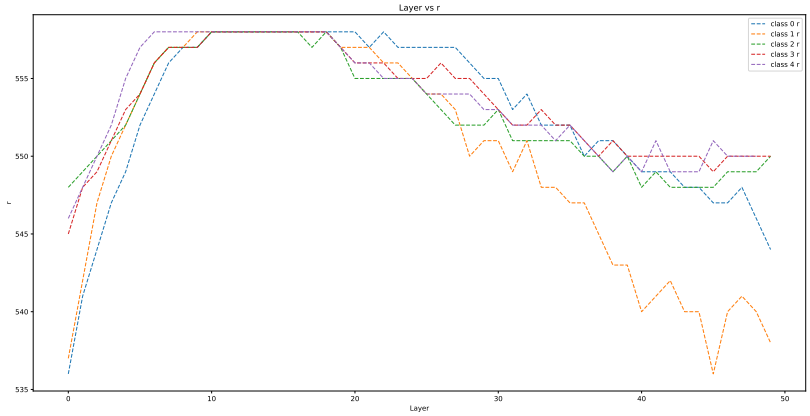
Experiments

We train a 50 layer ReduNet on 500 samples of 5 classes MNIST. We train the model with tolerance $\tau = 0.01$, learning rate $\eta = 0.1$, and ball packing error $\varepsilon^2 = 0.1$. We observe the following loss curves and number of eigenvectors chosen to approximate the class distributions at each layer (3.4).

Recall that the dimension of vectorized MNIST images is 784. With a tolerance of $\tau = 0.01$, each class chose approximately 550 vectors to approximate $\boldsymbol{C}_j$ at each layer.



(a) We plot the training loss over iterations using approximated $\boldsymbol{C}_j$'s and true $\boldsymbol{C}_j$'s.



(b) Number of eigenvectors r selected over iterations to approximate $\boldsymbol{C}_j$.

Figure 3.4: ReduNet trained on MNIST with 500 samples. Notice that the number of eigenvectors chosen is proportional to the compression loss

Additionally, we see that with further training, the number of eigenvectors r needed to approximate each class's compression matrix decreases. This is as expected, since at a local minima of the MCR^2 objective, the eigenvector of $\boldsymbol{Z}\boldsymbol{Z}^\top$ become closer to the eigenvectors of $\boldsymbol{Z}_j\boldsymbol{Z}_j^\top,\ \forall j$. However 550 vectors is no where close to $\frac{d}{k} = \frac{784}{5}$ which implies that that it is difficult to approximate $\boldsymbol{Z}_j$ using a subset of eigenvectors of $\boldsymbol{Z}$, especially at the initial layers. Additionally, this suggests it takes a while for Z to converge to an optimal solution.

3.3 Conclusion

From our experiments, we conclude that low rank approximation of the covariance matrices is an efficient way to reduce memory of ReduNet. By setting $\tau = 0.1$ we found a significant reduction in memory footprint in both our experiments in MNIST and CIFAR10 3.3. For future work, it would be interesting to see if we can identify low rank memory reduced networks for the Convolutional ReduNet [2], where $\boldsymbol{E}$ and $\boldsymbol{C}_j$ are circulant matrices. Additionally,

Bibliography

[1] P. Buzzega et al. "Dark Experience for General Continual Learning: a Strong, Simple Baseline". In: *Adv. Neural Inform. Process. Syst.* (2020).

[2] Ryan Chan et al. "Deep Networks from the Principle of Rate Reduction". In: *arXiv preprint arXiv:2010.14765* (2020).

[3] Kaiming He et al. "Deep residual learning for image recognition". In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2016).

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

[5] Ronald Kemker and Christopher Kanan. "Fearnet: Brain-inspired model for incremental learning". In: *arXiv preprint arXiv:1711.10563* (2017).

[6] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *PNAS* (2017).

[7] Alex Krizhevsky. "Learning multiple layers of features from tiny images". In: *Technical report* (2009).

[8] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[9] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann. lecun.com/exdb/mnist/* (1998).

[10] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[11] Xilai Li et al. "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting". In: *arXiv preprint arXiv:1904.00310* (2019).

[12] Zhizhong Li and Derek Hoiem. "Learning without Forgetting". In: *Eur. Conf. Comput. Vis.* (2016).

[13] Yi Ma et al. "Segmentation of multivariate mixed data via lossy data coding and compression". In: *IEEE transactions on pattern analysis and machine intelligence* 29.9 (2007), pp. 1546–1562.

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

[15] Vardan Papyan, X.Y. Han, and David Donoho. "Prevalence of Neural Collapse during the Terminal Phase of Deep Learning Training". In: *PNAS* (2020).

[16] Sylvestre-Alvise Rebuffi et al. "iCaRL: Incremental Classifier and Representation Learning". In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2017).

[17] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[18] Kate Saenko et al. "Adapting visual category models to new domains". In: *European conference on computer vision*. Springer. 2010, pp. 213–226.

[19] Jonathan Schwarz et al. "Progress & Compress: A scalable framework for continual learning". In: *ICML* (2018).

[20] Gido van de Ven and Andreas Tolias. "Three scenarios for continual learning". In: *Adv. Neural Inform. Process. Syst.* (2018).

[21] Chenshen Wu et al. "Memory replay gans: Learning to generate new categories without forgetting". In: *Advances in Neural Information Processing Systems*. 2018, pp. 5962–5972.

[22] Yue Wu et al. "Large scale incremental learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 374–382.

[23] Ziyang Wu et al. "Incremental learning via rate reduction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2021).

[24] Lu Yu et al. "Semantic Drift Compensation for Class-Incremental Learning". In: *IEEE Conf. Comput. Vis. Pattern Recog.* (2020).

[25] Yaodong Yu et al. "Learning Diverse and Discriminative Representations via the Principle of Maximal Coding Rate Reduction". In: *Adv. Neural Inform. Process. Syst.* (2020).

[26] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual Learning Through Synaptic Intelligence". In: *ICML* (2017).

[27] Junting Zhang et al. "Class-incremental learning via deep model consolidation". In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 1131–1140.