Incremental Learning via Rate Reduction

Ziyang Wu* Cornell zw287@cornell.edu
Christina Baek* UC Berkeley ke.baek@berkeley.edu
Chong You UC Berkeley cyou@berkeley.edu
Yi Ma UC Berkeley yima@eecs.berkeley.edu

Abstract

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.

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, 25], 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 utiliz-

* The first two authors contributed equally to this work.

ing 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. 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 [23, 26] 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*. [25] 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 4, after we have introduced our method properly.

3. Preliminaries

In this section, we provide a brief background on the principle of rate reduction and the "white box" network architecture (i.e., ReduNet) derived from such a principle.

3.1. Principle of 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 $\boldsymbol{y} = g(\boldsymbol{z}, \boldsymbol{w})$ for the downstream classification task. 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 labels. Therefore, it is unclear to what extent the feature representation captures any intrinsic structure of the data. Recent work [15] shows that this direct label fitting leads to a phenomena called *neural collapse*, where within-class variability and structural information are completely suppressed.

To address the aforementioned problem, a recent work by Yu *et al*. [24] presented a framework for learning useful and geometrically meaningful representation 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 the following *rate reduction*:

$$\Delta R(\boldsymbol{Z}) = R(\boldsymbol{Z}) - R_c(\boldsymbol{Z}, \boldsymbol{\Pi}), \tag{1}$$

subjecting to the constraint that $\boldsymbol{Z}$ is properly normalized, e.g., with the Frobenius norm of class features $\boldsymbol{Z}^j = \boldsymbol{Z}\boldsymbol{\Pi}^j$ to scale with the number of samples in class j: $\|\boldsymbol{Z}^j\|_F^2 = m_j = \text{tr}(\boldsymbol{\Pi}^j)$. In above, we denote

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

$$R_c(\boldsymbol{Z}, \boldsymbol{\Pi}) = \sum_{j=1}^{k}\frac{\gamma_j}{2}\log\det\left(\boldsymbol{I} + \alpha_j\boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^\top\right). \tag{3}$$

where $\alpha = d/(m\epsilon^2)$, $\alpha_j = d/(\text{tr}(\boldsymbol{\Pi}^j)\epsilon^2)$, $\gamma_j = \text{tr}(\boldsymbol{\Pi}^j)/m$, and $\epsilon > 0$ is a prescribed quantization error. $R(\boldsymbol{Z})$, 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. This coding rate measure utilizes local ϵ-ball packing to estimate the coding rate of the latent distribution from finite samples [13].

In [24], it is demonstrated empirically and theoretically that maximizing $\Delta R(\boldsymbol{Z})$ enforces the latent class distributions to be low-dimensional subspace-like distributions of approximately d/k dimension. In addition, these class distributions are orthogonal to each other. By doing so, the representation is between-class discriminative, whilst maintaining intra-class diversity. Moreover, these features have precise statistical and geometric interpretations.

3.2. Rate Reduction Network

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) is given by

$$\begin{aligned}\boldsymbol{Z}_{\ell+1} &\propto \boldsymbol{Z}_\ell + \eta\left(\frac{\partial\Delta R}{\partial\boldsymbol{Z}}\Big|_{\boldsymbol{Z}_\ell}\right)\\ &= \boldsymbol{Z}_\ell + \eta\Big(\boldsymbol{E}_\ell\boldsymbol{Z}_\ell - \sum_{j=1}^{k}\gamma_j\boldsymbol{C}_\ell^j\boldsymbol{Z}_\ell^j\Big)\\ \text{s.t.}\quad &\|\boldsymbol{Z}_{\ell+1}^j\|_F^2 = \text{tr}(\boldsymbol{\Pi}^j) = m_j \quad \forall j \in \{1, .., k\},\end{aligned} \tag{4}$$

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

$$\boldsymbol{E}_\ell = \alpha\left(\boldsymbol{I} + \alpha\boldsymbol{Z}_\ell\boldsymbol{Z}_\ell^\top\right)^{-1}, \tag{5}$$

$$\boldsymbol{C}_\ell^j = \alpha_j\left(\boldsymbol{I} + \alpha_j\boldsymbol{Z}_\ell^j\boldsymbol{Z}_\ell^{j\top}\right)^{-1}. \tag{6}$$

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.

Once the projected gradient ascent is completed, each gradient step can be interpreted as one layer of a neural

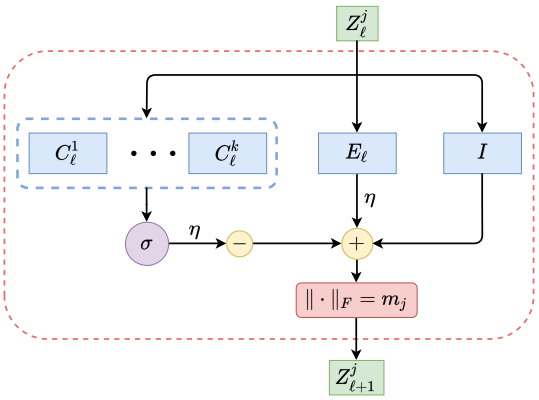

Figure 1. ReduNet Architecture in which we here adopt a slightly different normalization than [2] but is more suitable for the incremental learning as we will see in our derivation.

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. Then, given a test data $\boldsymbol{x} \in \mathbb{R}^d$, its feature can be computed by setting $\boldsymbol{z}_0 = \boldsymbol{x}$ and iteratively carrying out the following incremental transform

$$\boldsymbol{z}_{\ell+1} \propto \boldsymbol{z}_\ell + \eta\Big(\boldsymbol{E}_\ell \boldsymbol{z}_\ell - \sum_{j=1}^{k} \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \boldsymbol{\pi}^j(\boldsymbol{z}_\ell)\Big). \quad (7)$$

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 (7) by the following estimated membership

$$\hat{\boldsymbol{\pi}}_\ell^j(\boldsymbol{z}) = \frac{\exp\Big(-\lambda k \|\boldsymbol{C}_\ell^j \boldsymbol{z}\|\Big)}{\sum_{j=1}^k \exp\Big(-\lambda k \|\boldsymbol{C}_\ell^j \boldsymbol{z}\|\Big)} \in [0,1], \quad (8)$$

where $\lambda > 0$ is a confidence parameter. This leads to a nonlinear operator $\sigma\Big(\boldsymbol{C}_\ell^1 \boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k \boldsymbol{z}_\ell\Big) \doteq \sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \hat{\boldsymbol{\pi}}_\ell^j$ that, after being plugged into (7), produces a nonlinear layer as summarized in Figure 2. Stacking multiple such layers produces a multi-layer neural network for extracting discriminative features. Then, a nearest subspace classifier as the one presented in Section 5.3 can classify the data. In addition, each layer is interpretable and computed explicitly.

4. 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 $\{\boldsymbol{D}_i, i \le t\}$ were available.

4.1. 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 (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 3.2. 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

$$\begin{aligned}\boldsymbol{E}_0 &= \alpha\Big(\boldsymbol{I} + \alpha \boldsymbol{Z}_0 \boldsymbol{Z}_0^\top\Big)^{-1} \\ &= \alpha\Big(\boldsymbol{I} + \alpha\big(\boldsymbol{Z}_{0,t}\boldsymbol{Z}_{0,t}^\top + \boldsymbol{Z}_{0,t'}\boldsymbol{Z}_{0,t'}^\top\big)\Big)^{-1},\end{aligned} \quad (9)$$

and

$$\boldsymbol{C}_0^j = \begin{cases} \alpha_j\Big(\boldsymbol{I} + \alpha_j \boldsymbol{Z}_{0,t}^j \boldsymbol{Z}_{0,t}^{j\top}\Big)^{-1}, & \text{if } j \le k_t, \\ \alpha_j\Big(\boldsymbol{I} + \alpha_j \boldsymbol{Z}_{0,t'}^j \boldsymbol{Z}_{0,t'}^{j\top}\Big)^{-1}, & \text{else,} \end{cases} \quad (10)$$

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 $\mathbf{\Theta}_t$. Specifically, by denoting the compression matrices of $\mathbf{\Theta}_t$ as $\{\boldsymbol{C}^j_{\ell,t}\}$ for $\ell \in \{0, ..., L-1\}$, we have

$$\boldsymbol{Z}^j_{0,t}\boldsymbol{Z}^{j\top}_{0,t} = \left((\boldsymbol{C}^j_{0,t}/\alpha_j)^{-1} - \boldsymbol{I}\right)/\alpha_j. \tag{11}$$

Next, we show by induction that one can recursively compute $\boldsymbol{E}_\ell$ and $\{\boldsymbol{C}^j_\ell\}$ of $\mathbf{\Theta}$ for $\ell > 0$ from (9) and (10). To construct layer 1 of $\mathbf{\Theta}$, we observe that the output features of class j at layer 0 *before* normalization is as follows.

$$\boldsymbol{P}^j_0 = (\boldsymbol{Z}_0 + \eta\boldsymbol{E}_0\boldsymbol{Z}_0 - \eta\sum_{i=1}^{k}\gamma_i\boldsymbol{C}^i_0\boldsymbol{Z}^i_0)\mathbf{\Pi}^j \tag{12}$$

$$= \boldsymbol{Z}^j_0 + \eta\boldsymbol{E}_0\boldsymbol{Z}^j_0 - \eta\gamma_j\boldsymbol{C}^j_0\boldsymbol{Z}^j_0 \tag{13}$$

$$= \big(\underbrace{\boldsymbol{I} + \eta\boldsymbol{E}_0 - \eta\gamma_j\boldsymbol{C}^j_0}_{\boldsymbol{L}^j_0 \in \mathbb{R}^{d\times d}}\big)\boldsymbol{Z}^j_0. \tag{14}$$

Notice the term $\boldsymbol{L}^j_0$ only depends on quantities already obtained at layer 0. To compute $\boldsymbol{E}_1$ and $\boldsymbol{C}^j_1$, we need the covariance matrix of $\boldsymbol{P}^j_0$, which we observe to be

$$\boldsymbol{T}^j_1 = \boldsymbol{P}^j_0\boldsymbol{P}^{j\top}_0 = \boldsymbol{L}^j_0\boldsymbol{Z}^j_0\boldsymbol{Z}^{j\top}_0\boldsymbol{L}^{j\top}_0. \tag{15}$$

Notice that $\boldsymbol{T}^j_1$ can be expressed with known quantities of $\boldsymbol{L}^j_0$ and $\boldsymbol{Z}^j_{0,t}\boldsymbol{Z}^{j\top}_{0,t}$ if $j \le k_t$ or $\boldsymbol{Z}^j_{0,t'}\boldsymbol{Z}^{j\top}_{0,t'}$ if $j > k_t$. The remaining step would be to re-scale $\boldsymbol{T}^j_1$ as the updated representation $\boldsymbol{P}^j_0$ needs to be normalized to get $\boldsymbol{Z}^j_1$. Recall that we adopt the normalization scheme that imposes the Frobenius norm of each class $\boldsymbol{Z}_j$ to scale with m_j:

$$\|\boldsymbol{Z}^j_1\|^2_F = m_j \iff \mathrm{tr}\big(\boldsymbol{Z}^j_1\boldsymbol{Z}^{j\top}_1\big) = m_j. \tag{16}$$

The re-scaling factor is then easy to calculate:

$$\boldsymbol{Z}^j_1\boldsymbol{Z}^{j\top}_1 = \frac{m_j}{\mathrm{tr}(\boldsymbol{T}^j_1)}\boldsymbol{T}^j_1. \tag{17}$$

From above, we see that we can obtain the correct value of the covariance matrix $\boldsymbol{Z}^j_1\boldsymbol{Z}^{j\top}_1$, from which we can derive $\boldsymbol{E}_1$ and $\boldsymbol{C}^j_1$ for layer 1 of the joint network $\mathbf{\Theta}$ and obtain $\boldsymbol{Z}^j_{2,t'}$. With these values, we can compute $\boldsymbol{T}^j_2$.

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

$$\boldsymbol{Z}^j_\ell\boldsymbol{Z}^{j\top}_\ell = \frac{m_j}{\mathrm{tr}(\boldsymbol{T}^j_\ell)}\boldsymbol{T}^j_\ell. \tag{18}$$

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 $\boldsymbol{Z}^j_{\ell-1,t}\boldsymbol{Z}^{j\top}_{\ell-1,t} \in \mathbb{R}^{d\times d}$ 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. This procedure can be naturally extended to settings with more than two tasks.

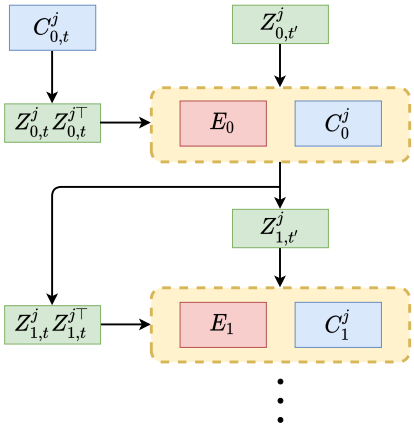

Figure 2. The joint network can be derived using simply $\boldsymbol{Z}_{0,t}\boldsymbol{Z}^\top_{0,t}$. We do not need the task t data $\boldsymbol{Z}_{0,t}$ directly.

Algorithm 1 Incremental Learning with ReduNet

Input: Network $\mathbf{\Theta}_t$ 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 $\mathbf{\Sigma}^j_{0,t} = \boldsymbol{Z}^j_{0,t}\boldsymbol{Z}^{j\top}_{0,t}, \forall j \in \{1, ..., k_t\}$ by (11).
for $\ell = 0, 1, 2, ..., L-1$ **do**
  $\mathbf{\Sigma}_{\ell,t} = \sum_{j=1}^{k_t}\mathbf{\Sigma}^j_{\ell,t}$,
  $\mathbf{\Sigma}_{\ell,t'} = \sum_{j=k_t+1}^{k_t+k_{t'}}\boldsymbol{Z}^j_{\ell,t'}\boldsymbol{Z}^{j\top}_{\ell,t'}$,
  $\boldsymbol{E}_\ell = \alpha\big(\boldsymbol{I} + \alpha(\mathbf{\Sigma}_{\ell,t} + \mathbf{\Sigma}_{\ell,t'})\big)^{-1}$,
  $\boldsymbol{L}^j_\ell = \boldsymbol{I} + \eta\boldsymbol{E}_\ell - \eta\gamma_j\boldsymbol{C}^j_\ell \quad \forall j \in \{1, ..., k_t\}$,
  for $j = 1, 2, ..., k_t$ **do**
    $\boldsymbol{C}^j_\ell = \alpha_j\big(\boldsymbol{I} + \alpha_j\mathbf{\Sigma}^j_\ell\big)^{-1}$,
    $\boldsymbol{T}^j_{\ell+1,t} = \boldsymbol{L}^j_\ell\mathbf{\Sigma}^j_{\ell,t}\boldsymbol{L}^{j\top}_\ell$,
    $\mathbf{\Sigma}^j_{\ell+1,t} = \frac{m_j}{\mathrm{tr}(\boldsymbol{T}^j_{\ell+1,t})}\boldsymbol{T}^j_{\ell+1,t}$.
  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 $\mathbf{\Theta}$ with parameters $\boldsymbol{E}_\ell$ and $\{\boldsymbol{C}^j_\ell\}$.

4.2. 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, 25] regularize the original weights of the model at each training session, effectively freezing cer-

tain parts of the 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 3, 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 $\boldsymbol{C}_\ell^j$ 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.

5. 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.

5.1. 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.

5.2. 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), 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 (7) with estimated labels computed as in (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 (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 $32 \times 32 \times 5$,

[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.

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 [25] 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.

5.3. 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 predicted by a nearest subspace classifier is

$$y = \underset{y \in 1,\ldots,k}{\arg\min} \left\| (\boldsymbol{I} - \boldsymbol{U}^y \boldsymbol{U}^{y\top}) \boldsymbol{z}_{test} \right\|_2^2, \tag{19}$$

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.

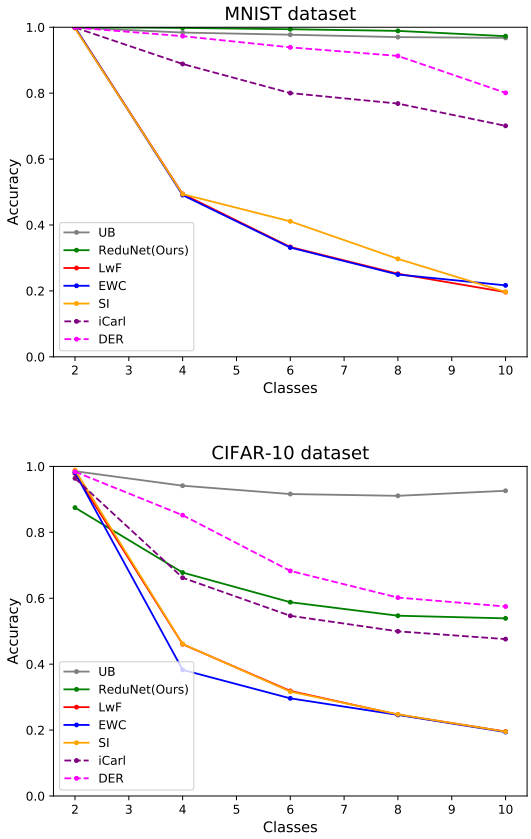

Figure 3. Incremental learning results (accuracy) on MNIST and CIFAR-10. Both datasets have 5 incremental batches. We also provide the upper bound (UB) given by joint training a model utilizing the same architecture as the baseline methods. In *solid* lines are regularization-based methods and in *dashed* 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.

5.4. 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 3. Additionally, we observe ReduNet shows significantly less forgetting (see Table 1).

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

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

On MNIST, we observe a 3% decay in accuracy across the tasks on ReduNet versus a 20-80% decay on benchmark methods (see Figure 3). 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 algorithms fail spectacularly due to catastrophic forgetting. In fact, as shown in Table 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, 23]. 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 3). 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 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.

6. Conclusions and Future Work

In this work, we have 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 want to emphasize that it is not the purpose of this work 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. 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.

Acknowledgement Yi Ma acknowledges support from ONR grant N00014-20-1-2002 and the joint Simons Foundation-NSFDMS grant #2031899.

References

[1] P. Buzzega, M. Boschini, A. Porrello, D. Abati, and S. Calderara. Dark experience for general continual learning: a strong, simple baseline. *Adv. Neural Inform. Process. Syst.*, 2020. 1, 5, 7, 8

[2] Ryan Chan, Yaodong Yu, Chong You, Haozhi Qi, John Wright, and Yi Ma. Deep networks from the principle of rate reduction. *arXiv preprint arXiv:2010.14765*, 2020. 1, 3, 4, 7

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016. 7

[4] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[5] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017. 2

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *PNAS*, 2017. 1, 2, 5

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Technical report*, 2009. 2

[8] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6

[9] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun.com/exdb/mnist/*, 1998. 2

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[11] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. *arXiv preprint arXiv:1904.00310*, 2019. 1, 2, 6

[12] Zhizhong Li and Derek Hoiem. Learning without forgetting. *Eur. Conf. Comput. Vis.*, 2016. 1, 2, 5, 7

[13] Yi Ma, Harm Derksen, Wei Hong, and John Wright. Segmentation of multivariate mixed data via lossy data coding and compression. *IEEE transactions on pattern analysis and machine intelligence*, 29(9):1546–1562, 2007. 3

[14] 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. 1

[15] Vardan Papyan, X.Y. Han, and David Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *PNAS*, 2020. 3

[16] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph Lampert. icarl: Incremental classifier and representation learning. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2017. 1, 2, 5, 7

[17] 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. 1, 2, 6

[18] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European conference on computer vision*, pages 213–226. Springer, 2010. 1

[19] Jonathan Schwarz, Jelena Luketina, Wojciech Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *ICML*, 2018. 1, 2, 5, 7

[20] Gido van de Ven and Andreas Tolias. Three scenarios for continual learning. *Adv. Neural Inform. Process. Syst.*, 2018. 2

[21] Chenshen Wu, Luis Herranz, Xialei Liu, Joost van de Weijer, Bogdan Raducanu, et al. Memory replay gans: Learning to generate new categories without forgetting. In *Advances in Neural Information Processing Systems*, pages 5962–5972, 2018. 2

[22] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019. 1, 2, 5

[23] Lu Yu, Bartomiej Twardowski, Xialei Liu, Luis Herranz, Kai Wang, Yongmei Cheng, Shangling Jui, and Joost van de Weijer. Semantic drift compensation for class-incremental learning. *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020. 2, 8

[24] Yaodong Yu, Ryan Chan, Chong You, Chaobing Song, and Yi Ma. Learning diverse and discriminative representations via the principle of maximal coding rate reduction. *Adv. Neural Inform. Process. Syst.*, 2020. 3

[25] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *ICML*, 2017. 1, 2, 5, 7

[26] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1131–1140, 2020. 2