GDumb: A Simple Approach that Questions Our Progress in Continual Learning

Ameya Prabhu[1] Philip H.S. Torr[1] Puneet K. Dokania[1,2]
{ameya,phst,puneet}@robots.ox.ac.uk

University of Oxford[1] & Five AI Ltd., UK[2]

Abstract. We discuss a general formulation for the Continual Learning (CL) problem for classification—a learning task where a stream provides samples to a learner and the goal of the learner, depending on the samples it receives, is to continually upgrade its knowledge about the old classes and learn new ones. Our formulation takes inspiration from the open-set recognition problem where test scenarios do not necessarily belong to the training distribution. We also discuss various quirks and assumptions encoded in recently proposed approaches for CL. We argue that some oversimplify the problem to an extent that leaves it with very little practical importance, and makes it extremely easy to perform well on. To validate this, we propose GDumb that (1) greedily stores samples in memory as they come and; (2) at test time, trains a model from scratch using samples only in the memory. We show that even though GDumb is not specifically designed for CL problems, it obtains state-of-the-art accuracies (often with large margins) in almost all the experiments when compared to a multitude of recently proposed algorithms. Surprisingly, it outperforms approaches in CL formulations for which they were specifically designed. This, we believe, raises concerns regarding our progress in CL for classification. Overall, we hope our formulation, characterizations and discussions will help in designing realistically useful CL algorithms, and GDumb will serve as a strong contender for the same.

1 Introduction

A fundamental characteristic of natural intelligence is its ability to continually learn new concepts while updating information about the old ones. Realizing that very objective in machines is precisely the motivation behind continual learning (CL). While current machine learning (ML) algorithms can achieve excellent performance given any single task, learning new (or even related) tasks continually is extremely difficult for them as, in such scenarios, they are prone to the phenomenon called catastrophic forgetting [1,2]. Significant attention has been paid recently to this problem [3,4,5,6,7,8] and a diverse set of approaches have been proposed in the literature (refer [9] for an overview). However, these approaches impose different sets of simplifying constraints to the CL problem and propose tailored algorithms for the same. Sometimes these constraints are so rigid that they even break the notion of learning continually, for example, one such

constraint would be knowing a priori the subset of labels a given input might take. In addition, these approaches are never tested exhaustively on useful scenarios. Keeping this observation in mind, we suggest that it is of paramount importance to understand the caveats in these simplifying assumptions, understand why these simplified forms are of little practical usability, and shift our focus on a more *general* and practically useful form of continual learning formulation to help progress the field.

To this end, we first provide a *general* formulation of CL for classification. Then, we investigate popular variants of existing CL algorithms, and categorize them based on the simplifying assumptions they impose over the said general formulation. We discuss how each of them impose constraints either over the growing nature of the label space, the size of the label space, or over the resources available. One of the primary drawbacks of these restricted settings is that algorithms tailored towards them fail miserably when exposed to a slightly different variant of CL, making them extremely specific to a particular situation. We would also like to emphasize that there is no explicit consensus among researchers regarding which formulation of CL is the most appropriate, leading to a diverse experimental scenarios, none of which actually mimic the general form of CL problem one would face when exposed to the real-world.

Then, we take a step back and design an extremely simple algorithm with almost no simplifying assumptions compared to the recent approaches. We call this approach GDumb (Greedy Sampler and Dumb Learner). As the name suggest, the two core components of our approach are a *greedy sampler* and a *dumb learner*. Given a memory budget, the sampler greedily stores samples from a data-stream while making sure that the classes are balanced, and, *at inference*, the learner (neural network) is trained from scratch (hence dumb) using all the samples stored in the memory. When tested on a variety of scenarios on which various recent works have proposed highly tuned algorithms, GDumb surprisingly provides state-of-the-art results with large margins in almost all the cases.

The fact that GDumb, even though not designed to handle the intricacies in the challenging CL problems, outperforms recently proposed algorithms in their own experimental set-ups, is alarming. It raises concerns relating to the popular and widely used assumptions, evaluation metrics, and also questions the efficacy of various recently proposed algorithms for continual learning.

2 Problem Formulation, Assumptions, and Trends

To provide a *general* and practically useful view of CL problem, we begin with the following example. Imagine a robot walking in a living room solving a task that requires it to identify all the objects it encounters. In this setting, the robot will be identifying known objects that it has learned in the past, will be learning about a few *unknown* objects by asking an oracle to provide labels for them, and, at the same time, will be updating its information about the known objects if the new instances of them provided extra cues useful for the task. In a nutshell,

the robot begins with some partial information about the world and keeps on improving its knowledge about it as it explores new parts of the world.

Inspired by this example, a realistic formulation of continual learning for classification would be where there is a stream of training samples or data accessible to a learner, each sample comprising a two-tuple $(\mathbf{x}_t, \mathbf{y}_t)$, where t represents the timestamp or the sample index. Let $\mathcal{Y}_t = \cup_{i=1}^{t} \mathbf{y}_i$ be the set of labels seen until time t, then it is trivial to note that $\mathcal{Y}_{t-1} \subseteq \mathcal{Y}_t$. This formulation implies that the stream might give us a sample that either belongs to a new class or to the old ones. Under this setting, at any *given* t, the objective is to provide a mapping $f_{\theta_t} : \mathbf{x} \rightarrow \mathbf{y}$ that can accurately map a sample $\mathbf{x}$ to a label $\mathbf{y} \in \mathcal{Y}_t \cup \bar{\mathbf{y}}$, where $\bar{\mathbf{y}}$ indicates that the sample does not belong to any of the learned classes. Notice, addition of this extra label $\bar{\mathbf{y}}$ assumes that while training, there is incomplete knowledge about the world and a test sample might come from outside the training distribution. Interestingly, it connects an instance of CL very well with the well known open-set classification problem [10]. However, in CL, the learner, with the help of an oracle (*e.g.*, active learning), could improve its knowledge about the world by learning the semantics of samples inferred as $\bar{\mathbf{y}}$.

2.1 Simplifying Assumptions in Continual Learning

The above discussed formulation is general in the sense that it does not put any constraints whatsoever on the growing nature of the label space, nature of test samples, and size of the output space $|\lim_{t\rightarrow\infty} \mathcal{Y}_t|$. It does not put any restrictions on the resources (compute and storage) one might pick to get a reliable mapping $f_{\theta_t}(.)$ either, however, the lack of information about the nature and the size of the output space makes the problem extremely hard. This has compelled almost all the work in this direction to impose additional simplifying constraints or assumptions. These assumptions are so varied that it is difficult to compare one CL algorithm with another as a slight variation in the assumption might change the complexity of the problem dramatically. For better understanding, below we discuss all the popular assumptions, highlight their drawbacks, and categorize various recently proposed CL algorithms depending on the simplifying assumptions they make. One assumption common to all is that the test samples always belong to the training distribution.

Disjoint task formulation: This formulation is being used in almost all the recent works [4,5,6,7,8] whereby the assumption made is that at a particular *duration* in time, the data-stream will provide samples specific to a task, in a pre-defined order of *tasks*, and the aim is to learn the mapping by learning each task at a time sequentially. In particular, let $\mathcal{Y} = \lim_{t\rightarrow\infty} \mathcal{Y}_t$ be the set of labels that the stream might present until it runs out of samples. Recall, in the general CL formulation, the size of $\mathcal{Y}$ is unknown and the samples can be presented in any order. This label space $\mathcal{Y}$ is then divided into different disjoint subsets (could be a random or an informed split), where each label subset $\mathcal{Y}_i$ represents a task and the *sharp* transition between these sets is called *task boundaries*. Let there be m splits (typically the split is balanced with nearly equal number of

classes) then $\mathcal{Y} = \cup_i^m \mathcal{Y}_i$, and $\mathcal{Y}_i \cap \mathcal{Y}_j = \emptyset, \forall i \neq j$. An easy and widely used example is to divide ten digits of MNIST into 5 disjoint tasks where each task comprises of the samples from two consecutive digits and the stream is *controlled* to provide samples for each task in a pre-defined order, say $\{0, 1\}, \cdots, \{8, 9\}$. This formulation simplifies the general CL problem to a great extent as the unknown growing nature of the label space is now being restricted and is known. It provides a very strong prior to the learner and helps in deciding both the space budget and the family of functions $f_\theta(.)$ to learn.

***Task-incremental** v/s **Class-incremental**:* To further make the training and the inference easier, a popular choice of CL formulation is the task-incremental continual learning (TI-CL) [7] where, along with the disjoint task assumption, the task information (or id) is also passed by an oracle during training and inference. Thus, instead of a two-tuple, a three-tuple $(\mathbf{x}, \mathbf{y}, \alpha)$ is given where $\alpha \in \mathbb{N}$ represents the task identifier. This formulation is also known as *multi-head* and is an extremely simplified form of the continual learning problem [8]. For instance, in the above mentioned MNIST example, at inference, if the input is $(\mathbf{x}, \alpha = 3)$, it implies that the sample either belongs to class 4, or to 5. Knowing this subset of labels a-prior dramatically reduces the label space during training and inference, and is relatively impractical to know in real-world scenarios. Whereas, in a class-incremental formulation (CI-CL) [4,8], also known as the *single-head*, we do not have any such information about the task id.

***Online CL** v/s **Offline CL**:* Note, the disjoint task formulation placed a restriction on the growing nature of the label space and inherently restricted the size of it, however, it did not put any constraints on the learner itself. Therefore, the learning paradigm may store task-specific samples coming from the stream depending on the space budget and then use them to update the parameters. Under this setting, in the *online* CL formulation, even though the learner is allowed to store samples as they come, they are not allowed to use a sample more than once for parameter update. Thus, the learner can not use the same sample (unless it is in the memory) multiple times at different iterations of the learning process. In contrast, *offline* CL allows unrestricted access to the *entire* dataset corresponding to a particular task (not to the previous ones) and one can use this dataset to learn the mapping by revisiting the samples again and again while performing multiple passes over the data [4].

***Memory based CL**:* As mentioned earlier, we only have access to all/subset of samples corresponding to the *current* task. This restriction makes it extremely hard for the model to perform well, in particular, on CI-CL setting as the absence of samples from the previous tasks makes it difficult to learn to distinguish samples from the current and the previous tasks due to catastrophic forgetting. Very little forgetting is normally observed in TI-CL as the given task identifier works as the indicator of task boundary, thus the model does not have to learn to differentiate labels among tasks. To reduce forgetting, a common practice, inspired by the complementary learning systems theory [38,39], is to store a subset

Form.	CI-CL	Online	Disjoint	Papers	Regularize	Memory	Distill	Param iso
A	✓	✓	✓	MIR[11], GMED[12]	×	✓	×	×
B	✓	×	✓	LwM[13], DMC[14]	×	×	✓	×
				SDC [15]	✓	×	×	×
				BiC[16], iCARL[4] UCIR[17], EEIL[18] IL2M[19], WA[20] PODNet[21], MCIL[22]	×	✓	✓	×
				RPS-Net[23], iTAML[24]	×	✓	✓	✓
				CGATE[25]	×	✓	×	✓
				RWALK[8]	✓	✓	×	×
C	×	×	✓	PNN[26], DEN[27]	×	×	×	✓
				DGR [28]	×	✓	×	×
				LwF[3]	×	×	✓	×
				P&C[29]	×	×	✓	✓
				APD[30]	✓	×	×	✓
				VCL[31]	✓	✓	×	×
				MAS[32], IMM[33] SI[5], Online-EWC[29] EWC[6]	✓	×	×	×
D	×	✓	✓	TinyER[34], HAL[35]	×	✓	×	×
				GEM[7], AGEM[36]	✓	✓	×	×
E	✓	✓	×	GSS[37]	×	✓	×	×

Table 1. Here we categorize various recently proposed CL approaches depending on the underlying simplifying assumptions they impose.

of samples from each task and use them while training on the current task. There primarily are two components under this setting: a *learner* and a *memorizer* (or sampler). The learner has the goal of obtaining representations which generalize beyond current task. The memorizer, on the other hand, deals with remembering (storing) a collection of episode-like memories from the previous tasks. In recent approaches [4,7,8], the learner is modeled by a neural network and the memorizer is modeled by memory slots which store samples previously encountered.

2.2 Recent Trends in Continual Learning

Typically, continual learning approaches are categorized by ways they tackle forgetting such as (1) regularization-based, (2) replay (or memory)-based, (3) distillation-based, and (4) parameter-isolation based (for details refer [9]). However, they do vary in terms of simplifying assumptions they encode, and we argue that keeping track of these assumptions is extremely important for fair comparisons, and also to understand the limitations of each of them. Since all these

Algorithm 1 Greedy Balancing Sampler

Init: counter C_0={}, $\mathcal{D}_0$={} with capacity k. Online samples arrive from t=1

function SAMPLE(x_t, y_t, $\mathcal{D}_{t-1}, \mathcal{Y}_{t-1}$) ▷ Input: New sample and past state
  $k_c = \frac{k}{|\mathcal{Y}_{t-1}|}$
  if $y_t \notin \mathcal{Y}_{t-1}$ or $C_{t-1}[y_t] < k_c$ **then**
    if $\sum_i C_i >= k$ **then** ▷ If memory is full, replace
      $y_r = argmax(C_{t-1})$ ▷ Select largest class, break ties randomly
      $(x_i, y_i) = \mathcal{D}_{t-1}$.random($y_r$) ▷ Select random sample from class y_r
      $\mathcal{D}_t = (\mathcal{D}_{t-1} - (x_i, y_i)) \cup (x_t, y_t)$
      $C_t[y_r] = C_{t-1}[y_r] - 1$
    else ▷ If memory has space, add
      $\mathcal{D}_t = \mathcal{D}_{t-1} \cup (x_t, y_t)$
    end if
    $\mathcal{Y}_t = \mathcal{Y}_{t-1} \cup y_t$
    $C_t[y_t] = C_{t-1}[y_t] + 1$
  end if
  return $\mathcal{D}_t$
end function

algorithms in some sense use combinations of the above discussed simplifying assumptions, to give a bird's eye view over all the recently proposed approaches, we categorize them in Table 1 depending on the simplifying assumptions they make. For example, Table 1 indicates that RWalk [8] is an approach designed for a CL formulation that is offline, class-incremental, and assumes sharp task boundaries. Algorithmically, it is regularization based and uses memory. Note, one can potentially modify these approaches to apply to other settings as well. For example, the same RWalk can also be used without memory, or can be applied on task-incremental offline formulation. However, we focus on the formulation these methods were originally designed for. We now discuss some high-level problems associated with the simplifying assumptions these approaches make.

Most models, metrics, classifiers, and samplers for CL inherently encode disjoint task (or sharp task boundary) assumption into their design, hence fail to generalize even with slight deviation from the this formulation. Similarly, popular metrics like forgetting and intransigence [7,8] are designed with this specific formulation encoded in their formal definition, and break with simple modifications like blurry boundaries (class-based, instead of sample-based, definitions of forgetting would appear as classes mix because of blurred boundaries).

Moving to TI-CL v/s CI-CL, these are two extreme cases where CI-CL (single-head) faces scaling issues as there is no restriction on the size of $|\lim_{t\to\infty} \mathcal{Y}_t|$, and TI-CL (multi-head) imposes a fixed, coherent two-level hierarchy among classes with oracle labels. This formulation is unrealistic in the sense that it does not allow dynamic contexts [40].

Lastly, Offline CL v/s Online CL is normally defined depending on whether an algorithm is allowed to revisit a sample repeatedly (unless it is in the memory) during the training process or not. The intention here is to make the continual

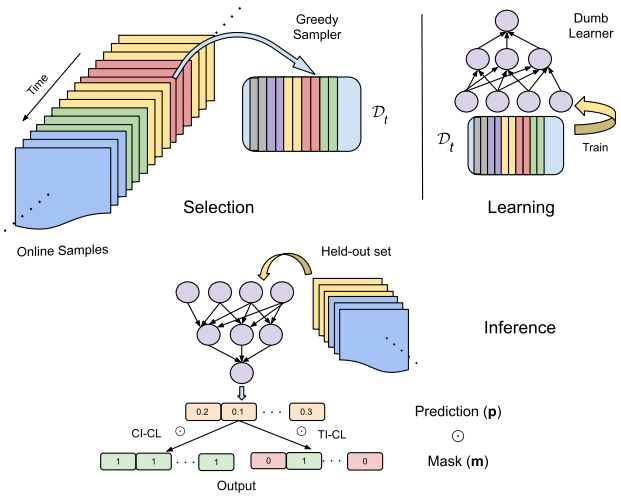

Fig. 1. Our approach (GDumb): The sampler greedily stores samples while balancing the classes. When asked, the learner trains a network from scratch on memory $\mathcal{D}_t$ provided by the sampler. If a mask $\mathbf{m}$ is given at inference, GDumb classifies on the subset of labels provided by the mask. Depending on the mask, GDumb's inference can vary between two extremes: CI (class-incremental) and TI (task-incremental) formulations.

learning algorithm fast enough so that it can learn *quickly* from a single (or few) sample without having the need of revisiting it. This distinction makes sense if we imagine a data stream spitting samples very fast, then the learner has to adapt itself very quickly. Therefore, the algorithm must provide an acceptable trade-off between space (number of samples to store) and time (training complexity) budgets. However, because of the lack of proper definition and evaluation schemes, there are algorithms doing very well on one end (use a sample only once), however, performing very poorly on the other end (very expensive learning process). For example, GEM [7], a widely known online CL algorithm, uses a sample only once, however, solves a quadratic program for parameter updates which is very time consuming. Therefore, without proper metrics or procedures to quantify how well various CL algorithms balance both space and time complexities, categorizing them into offline vs online might lead to wrong conclusions.

3 Greedy Sampler and Dumb Learner (GDumb)

We now propose a simple approach that does not put any restrictions, as discussed above, over the growing nature of the label space, task boundaries, online vs offline, and the ordering of the samples in which the data-stream provides them. Thus, can easily be applied to all the CL formulations discussed in Table 1. The only requirement is to be allowed to store some episodic memories. We emphasize that we do not claim that our approach solves the general CL problem.

Form.	Designed in	Model (Dataset)	memory (k)	Metric	CI-CL	Online	Disjoint
A1	[11]	MLP-400 (MNIST); ResNet18 (CIFAR10)	300, 500; 200, 500, 1000	Acc. (at end)			
A2	[12]	MLP-400 (MNIST); ResNet18 (CIFAR10)	500; 500	Acc. (at end)	✓	✓	✓
A3	[41]	MLP-400 (MNIST); ResNet18 (CIFAR10)	500; 1000	Acc. (at end)			
B1	[42]; [23]	MLP-400 (MNIST); ResNet18 (SVHN)	4400	Acc. (at end)			
B2	[4]	ResNet32 (CIFAR100)	2000	Acc. (avg in t)	✓	×	✓
B3	[21]	ResNet32 (CIFAR100); ResNet18 (ImageNet100)	1000-2000 (+20) x50	Acc. (avg in t)			
C1	[42]	MLP-400 (MNIST)	4400	Acc. (at end)	×	×	✓
C2	[9]	Many (TinyImageNet)	4500,9000	Acc. (at end)			
D	[36]	ResNet-18-S (CIFAR10)	0-1105 (+65) x17	Acc. (at end)	×	✓	✓
E	[37]	MLP-100 (MNIST); ResNet-18 (CIFAR10)	300; 500	Acc. (at end)	✓	✓	×

Table 2. Various CL formulations we considered in this work to evaluate GDumb. These formulations differ in terms of simplifying assumptions (refer Table 1) and also in terms of resources used. We ensure that selected benchmarks are diverse, covering all popular categorizations. Note, in B3 and D, memory is not constant– it increases over tasks uniformly by (+size) for xtasks times.

Rather, we experimentally show that our simple approach, that does not encode anything specific to the challenging CL problem at hand, is surprisingly effective compared to other approaches over all the formulations discussed previously, and also exposes important shortcomings with recent formulations and algorithms.

As illustrated in Figure 1, our approach comprises of two key components: a *greedy balancing sampler* and a *learner*. Given a memory budget, say k samples, the sampler greedily stores samples from the data-stream (max k samples) with the constraint to asymptotically balance class distribution (Algorithm 1). It is greedy in the sense that whenever it encounters a new class, the sampler simply creates a new bucket for that class and starts removing samples from the old ones, in particular, from the one with a maximum number of samples. Any tie is broken randomly, and a sample is also removed randomly assuming that each sample is equally important. Note, this sampler does not rely on task boundaries or any information about the number of samples in each class.

Let the set of samples greedily stored by the sampler in the memory at any instant in time be $\mathcal{D}_t$ (a dataset with $\leq k$ samples). Then, the objective of the learner, a deep neural network in our experiments, is to learn a mapping $f_{\theta_t} : \mathbf{x} \rightarrow \mathbf{y}$, where $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_t$. This way, using a small dataset that the sampler has stored, the learner learns to classify all the labels seen until time t. Let $\mathcal{Y}_t$ represents the set of labels in $\mathcal{D}_t$. Then, at inference, given a sample $\mathbf{x}$, the

Method	MNIST	
k	(300)	(500)
MLP-100		
FSS-Clust [37]	75.8 ± 1.7	83.4 ± 2.6
GSS-Clust [37]	75.7 ± 2.2	83.9 ± 1.6
GSS-IQP [37]	75.9 ± 2.5	84.1 ± 2.4
GSS-Greedy [37]	82.6 ± 2.9	84.8 ± 1.8
GDumb (Ours)	**88.9 ± 0.6**	**90.0 ± 0.4**
MLP-400		
GEN [43]	-	75.5 ± 1.3
GEN-MIR [11]	-	81.6 ± 0.9
ER [44]	-	82.1 ± 1.5
GEM [7]	-	86.3 ± 1.4
ER-MIR [11]	-	87.6 ± 0.7
GDumb (Ours)	-	**91.9 ± 0.5**

(A1)

Method	CIFAR10		
k	(200)	(500)	(1000)
GEM [7]	16.8 ± 1.1	17.1 ± 1.0	17.5 ± 1.6
iCARL [4]	28.6 ± 1.2	33.7 ± 1.6	32.4 ± 2.1
ER [44]	27.5 ± 1.2	33.1 ± 1.7	41.3 ± 1.9
ER-MIR [11]	29.8 ± 1.1	40.0 ± 1.1	47.6 ± 1.1
ER5 [11]	-	-	42.4 ± 1.1
ER-MIR5 [11]	-	-	49.3 ± 0.1
GDumb (Ours)	**35.0 ± 0.6**	**45.8 ± 0.9**	**61.3 ± 1.7**

(A1)

Table 3. (CI-Online-Disjoint) Performance on formulation A1.

prediction is made as

$$\hat{y} = \arg\max \ \mathbf{p} \odot \mathbf{m}, \tag{1}$$

where, $\mathbf{p}$ is the softmax probabilities over all the classes in $\mathcal{Y}_t$, $\mathbf{m} \in \{0,1\}^{|\mathcal{Y}_t|}$ is a user-defined mask, and $\odot$ denotes the Hadamard product. Note, our prediction procedure allows us to mask any combination of labels at inference. When $\mathbf{m}$ consists of all ones, the inference is exactly the same as that of *single-head* or class-incremental, and when the masking is done depending on the subset of classes in a particular task, it is exactly the same as *multi-head* or task-incremental. Since our sampler does not put any restrictions on the flow of the samples from the data-stream, and our learner does not require any task-boundaries, our overall approach puts minimal restrictions on the general continual learning problem. We would also like to emphasize that we do not use the class $\bar{\mathbf{y}}$ as discussed in our general formulation in Section 2, we leave that for future work. However, our objective does encapsulate all the recently proposed CL formulations with minimal possible assumptions, allowing us to provide a fair comparison.

4 Experiments

We now compare GDumb with various existing algorithms for several recently proposed CL formulations. As shown in Table 1, there broadly are five such formulations $\{A, B, \cdots, E\}$. Since even within a formulation there can be sub-categories depending on the resources used, we further enumerate them and present a more detailed categorization, keeping fair comparisons in mind, in Table 2. For example, B1 and B2 belong to the same formulation B, however,

they differ in terms of architectures, datsets, and memory sizes used in their respective papers. Therefore, in total, we pick 10 different formulations, most of them having multiple architectures and datasets (refer Appendix B for details).

Implementation details: GDumb uses the same fixed training settings, with *no* hyperparameter tuning whatsoever, in all the CL formulations. This is possible because of the fact that GDumb does not impose any simplifying assumptions. All results measure accuracy (fraction of correctly classifications) evaluated on the held-out test set. For all the formulations, GDumb uses an SGD optimizer, fixed batch size of 16, learning rates $[0.05, 0.0005]$, an SGDR [45] schedule with T_0= 1, T_{mult}= 2 and warm start of 1 epoch. Early stopping with patience of 1 cycle of SGDR, along with standard data augmentation is used. GDumb uses cutmix [46] with p=0.5 and α=1.0 for regularization on all datasets except MNIST. The training set-up comprises of an Intel i7 4790, 32GB RAM and a single GTX 1070 GPU. All results are averaged over 3 random seeds, each with different class-to-task assignments. In formulations B2 and B3, we strictly follow class order specified in iCARL [4] and PODNet [21]. Our pytorch implementation is publicly available at: https://github.com/drimpossible/GDumb.

4.1 Results and Discussions

Class Incremental Online CL with Disjoint Tasks (Form. A): The first sub-category under this formulation is A1, which follows exactly the same setting, on Split-MNIST and Split-CIFAR10, as presented in MIR [11]. Results are shown in Table 3. We observe that on both MNIST and CIFAR10 for all choices of k, GDumb outperforms all the approaches by a large margin. For example, in the case of MNIST with memory size $k = 500$, GDumb outperforms ER-MIR [11] by around 4.3%. Similarly, on CIFAR10 with memory sizes of 200, 500, and 1K, our approach outperforms current approaches by nearly 5%, 6% and 11%, respectively, convincingly achieving state-of-the-art results. Note, increasing the memory size from 200 to 1K in CIFAR10 increases the performance of GDumb by 26.3% (expected as GDumb is trained only on memory), whereas, this increase is only 18% in the case of ER-MIR [11]. Similar or even much worst improvements are noticed in other recent approaches, suggesting they might not be utilizing the memory samples efficiently.

We now benchmark our approach on Split-MNIST and Split-CIFAR10 as detailed by parallel works GMED [12] (sub-category A2) and ARM [41] (sub-category A3). We present results in Table 4 and show that GDumb outperforms parallel works like HAL [35], QMED [12], ARM [41] in addition to outperforming recent works GSS [37], MIR [11], and ADI [47], consistently across datasets. It outperforms the best alternatives in QMED [12] by over 4% and 10% on MNIST and CIFAR10, respectively, and in ARM [41] by over 5% and 13% on MNIST and CIFAR10 datasets, respectively. Results from ARM [41] indicate—(i) GDumb consistently outperforms other experience replay approaches and (ii) experience replay methods obtain much better performance than generative replay with much smaller memory footprint.

Method k	MNIST (500)	CIFAR-10 (500)
Fine tuning	18.8 ± 0.6	18.5 ± 0.2
AGEM [36]	29.0 ± 5.3	18.5 ± 0.6
BGD [48]	13.5 ± 5.1	18.2 ± 0.5
GEM [7]	87.2 ± 1.3	20.1 ± 1.4
GSS-Greedy [37]	84.2 ± 2.6	28.0 ± 1.3
HAL [35]	77.9 ± 4.2	32.1 ± 1.5
ER [44]	81.0 ± 2.3	33.3 ± 1.5
MIR [11]	84.9 ± 1.7	34.5 ± 2.0
GMED (ER) [12]	82.7 ± 2.1	35.0 ± 1.5
GMED (MIR) [12]	87.9 ± 1.1	35.5 ± 1.9
GDumb (Ours)	**91.9 ± 0.5**	**45.8 ± 0.9**

(A2)

Method	MNIST		CIFAR10	
	Memory	Accuracy	Memory	Accuracy
Finetune	0	18.8 ± 0.5	0	15.0 ± 3.1
GEN [28]	4.58	79.3 ± 0.6	34.5	15.3 ± 0.5
GEN-MIR [11]	4.31	82.1 ± 0.3	38.0	15.3 ± 1.2
LwF [3]	1.91	33.3 ± 2.5	4.38	19.2 ± 0.3
ADI [47]	1.91	55.4 ± 2.6	4.38	24.8 ± 0.9
ARM [41]	1.91	56.2 ± 3.5	4.38	26.4 ± 1.2
ER [44]	0.39	83.2 ± 1.9	3.07	41.3 ± 1.9
ER-MIR [11]	0.39	85.6 ± 2.0	3.07	47.6 ± 1.1
iCarl [4] (5 iter)	-	-	3.07	32.4 ± 2.1
GEM [7]	0.39	86.3 ± 0.1	3.07	17.5 ± 1.6
GDumb (ours)	**0.39**	**91.9 ± 0.5**	**3.07**	**61.3 ± 1.7**

(A3)

Table 4. (CI-Online-Disjoint) Performance on formulations A2 (left) and A3 (right).

Class Incremental Offline CL with Disjoint Tasks (Form. B): We proceed next to offline CI-CL formulations. We first compare our proposed approach with 12 popular methods on sub-category B1. Results are presented in Table 5 (left). Our approach outperforms all memory-based methods like GEM, RtF, DGR by over 5% on MNIST. We outperform the recent RPS-Net [23] and OvA-INN [49] by over 1%, and are as good as iTAML [24], on MNIST. On SVHN, we outperform recently proposed methods like RPS-Net by 4.5% and far exceeding methods like GEM. Note, we achieve the same accuracy as the best offline CL method iTAML [24] despite using an extremely simple approach in online fashion.

We now discuss two very interesting sub-categories B2 (as in iCARL [4]) and B3 (from a very recent work PODNet [21]). The primary difference between B2 and B3 merely lies in the number of classes per task. However, as will be seen, this minor difference changes the complexity of the problem dramatically. In the case of B2, CIFAR100 is divided into 20 tasks, whereas, B3 starts with a network trained on 50 classes and then learns one class per task incrementally (leading to 50 new tasks). Performance of GDumb on B2 and B3 formulations are shown in Table 5 (center) and Table 5 (right), respectively. An interesting observation here is that GDumb which performed nearly 20% worse than BiC and iCARL in B2, performs over 10-15% better than BiC, UCIR and iCARL in B3. This drastic shift against previous results might suggest that having higher number of classes per task and less number of tasks might give added advantage to scaling/bias correction type approaches, which otherwise would quickly deteriorate over greater timesteps. Furthermore, we note that our simple baseline narrowly outperforms PODNet (CNN) on both CIFAR100 and ImageNet100 datasets.

Task Incremental Offline CL with Disjoint Tasks (Form. C): We now proceed to compare the performance of GDumb in task incremental formulation.

Method	MNIST	SVHN
No memory		
MAS [32]	19.5 ± 0.3	17.3
SI [5]	19.7 ± 0.1	17.3
EWC [6]	19.8 ± 0.1	18.2
Online EWC [29]	19.8 ± 0.04	18.5
LwF [3]	24.2 ± 0.3	-
k=4400		
DGR [28]	91.2 ± 0.3	-
DGR+Distill	91.8 ± 0.3	-
GEM [7]	92.2 ± 0.1	75.6
RtF [50]	92.6 ± 0.2	-
RPS-Net [23]	96.2	88.9
OvA-INN [49]	96.4	-
iTAML [24]	97.9	**94.0**
GDumb (Ours)	**97.8** ± **0.2**	93.4 ± 0.4

(B1)

Method	CIFAR100	
	Acc. (Avg)	Acc. (last)
No memory		
Finetune	17.8 ± 0.72	5.9 ± 0.15
SI [5]	23.6 ± 1.90	13.3 ± 1.14
MAS [32]	24.7 ± 1.76	10.4 ± 0.80
EWC [6]	25.4 ± 1.99	9.5 ± 0.83
RWALK [8]	25.6 ± 1.92	11.1 ± 2.14
LwF [3]	32.3 ± 1.92	14.1 ± 0.87
DMC [14]	45.0 ± 1.96	23.8 ± 1.90
k=2000		
GDumb (Ours)	45.2 ± 1.70	24.1 ± 0.97
DMC++ [14]	56.8 ± 0.86	-
iCARL [4]	58.8 ± 1.90	42.9 ± 0.79
EEIL [18]	63.4 ± 1.6	-
BiC [16]	63.8	46.9

(B2)

Method	CIFAR100	ImageNet100
iCaRL [4]	44.2 ± 1.0	54.97
BiC [16]	47.1 ± 1.5	46.49
UCIR (NME) [17]	48.6 ± 0.4	55.44
UCIR (CNN) [17]	49.3 ± 0.3	57.25
PODNet (CNN) [21]	58.0 ± 0.5	62.08
GDumb (CNN)	58.4 ± 0.8	**62.86**
PODNet (NME) [21]	**61.4** ± **0.7**	-

(B3)

Table 5. (CI-Offline-Disjoint) Performance on B1, B2, and B3.

Recall, GDumb does not put any restrictions such as task vs class incremental, or online vs offline. However, in the case of GDumb, we use masking (subset of labels in a task) over softmax probabilities at test time to mimic TI-CL formulation.

Table 6 (left) shows the results on C1, a very widely used and most popular offline TI-CL (or multi-head) formulation for CL on Split-MNIST. We now move to C2 on Split-TinyImagenet (offline TI-CL formulation as in [9]). Note, in this particular formulation we used a different architecture called DenseNet-100-BC [54]. Results are presented in Table 6 (middle). We observe that for $k = 9000$, we outperform all 10 approaches including GEM and iCARL by margins of atleast 7%. When the memory is halved to $k = 4500$, we perform slightly better than GEM and nearly 3% worse than iCARL. Since we used different architecture, we do not claim that we would notice similar improvements had we trained GDumb using the networks used in respective papers. However, these results are still encouraging as the approaches we compare against are trained in TI-CL manner and GDumb is always trained in CI-CL manner (much more difficult).

Task Incremental Online CL with Disjoint Tasks (Form. D): We now compare GDumb with 12 TI-CL tuned online approaches with small memory (not a favourable setting for GDumb as it relies totally on the samples in the memory) and detail the results in Table 6 (right). We observe that GDumb outperforms 8 out of 11 approaches even though it is trained in CI-CL manner.

Class Incremental Online CL with Joint Tasks (Form. E): We now measure impact of imbalanced data stream with blurry task boundaries [37]. Results are presented in Table 7 (left). We outperform competing models by over 10% and 16%, overwhelming surpassing complicated methods attuned to this

Method (k)	MNIST (4400)
GEM [7]	98.42 ± 0.10
EWC [6]	98.64 ± 0.22
SI [5]	99.09 ± 0.15
Online EWC [29]	99.12 ± 0.11
MAS [32]	99.22 ± 0.21
DGR [28]	99.50 ± 0.03
LwF [3]	99.60 ± 0.03
DGR+Distil [28]	99.61 ± 0.02
RtF	99.66 ± 0.03
GDumb	**99.77 ± 0.03**

(C1)

Method	Parameters	Regularization	Accuracy
No stored samples			
mean-IMM [33]	3.5M	none	32.42
mode-IMM [33]	9.0M	dropout	42.41
SI [5]	3.5M/9.0M	L2/dropout	43.74
HAT [51]	3.5M/9.0M	L2	44.19
EWC [6]	613K	none	45.13
LwF [3]	9.0M	L2	48.11
EBLL [52]	9.0M	L2	48.17
MAS [32]	3.5M/9.0M	none	48.98
PackNet [53]	613K/3.5M	L2/dropout	55.96
k=4500			
GEM [7]	613K/3.5M	none/dropout	44.23
GDumb	834K	cutmix	45.50
iCARL [4]	613K/3.5M	dropout	48.55
k=9000			
GEM [7]	613K/3.5M	none/dropout	45.27
iCARL [4]	613K/3.5M	dropout	49.94
GDumb	834K	cutmix	**57.27**

(C2)

Method (k)	CIFAR100 (1105)
RWalk [8]	40.9 ± 3.97
EWC [6]	42.4 ± 3.02
Base	42.9 ± 2.07
MAS [32]	44.2 ± 2.39
SI [5]	47.1 ± 4.41
iCARL [4]	50.1
S-GEM [36]	56.2
PNN [26]	59.2 ± 0.85
GEM [7]	61.2 ± 0.78
A-GEM [36]	63.1 ± 1.24
TinyER [34]	68.5 ± 0.65
GDumb	60.3 ± 0.85

(D)

Table 6. (TI-Offline-Disjoint) Performance on C1 (left) and C2 (middle). (TI-Onlione-Disjoint) Performance on D (right).

benchmark. This demonstrates that GDumb works well even when almost all simplifying assumptions are removed.

4.2 Resources Needed

It is important that our approach is in the ballpark of online continual learning constraints of memory and compute usage to achieve its performance. We benchmark our resource consumption against the efficient CL algorithms in Table 7 (right) benchmarked with a V100 GPU on formulation E [36]. We observe that we require only 60s on a slower GTX 1070 GPU (and 350s on a 4790 i7 CPU), performing several times efficient than various recently proposed algorithm. Note that sampling time is negligible, while testing time is not included in the above.

4.3 Potential Future Extensions

Active Sampling: Given an importance value $v_t \in \mathrm{R}^+$ (by active learner) along with sample (x_t, y_t) at time t, we can extend our sampler by having the objective of storing most important samples (maximizing $\sum_{i=1}^{|\mathcal{D}_c|} v_i$) for any given class c in its storage of size k. This will allow an algorithm to reject less important samples. Of course, it is not clear how to learn to quantify *importance* of a sample.

Dynamic Probabilistic Masking: It is possible to extend masking in GDumb beyond CI-CL and TI-CL to dynamic task hierarchies across video/scene

Method	MNIST	CIFAR10
Reservoir [43]	69.12	-
GSS-Clust [37]	-	25.0
FSS-Clust [37]	-	26.0
GSS-IQP [37]	76.49	29.6
GSS-Greedy [37]	77.96	29.6
GDumb (Ours)	**88.93**	**45.8**

(E)

Method	Train time	Memory (Train)	Memory (Test)
Base	105s	P + B*H	P + B*H
EWC	250s	4*P + B*H	P + B*H
PNN	409s	2*P*T + B*H*T	2*P*T + B*H*T
GEM	5238s	P*T + (B+M)*H	P + B*H
A-GEM	449s	2*P + (B+M)*H	P + B*H
GDumb	**60s**	**P + M*H**	**P + B*H**

(Resources)

Table 7. (CI-Online-Joint) Performance on E (left). Note, this is particularly challenging as the tasks here are non-disjoint (blurry task boundary) with class-imbalance. On the (right), we benchmark resource consumption in terms of training time and memory usage. Memory cost is provided in terms of the total parameters P, the size of the minibatch B, the total size of the network hidden state H (assuming all methods use the same architecture), the size of the episodic memory M per task. GDumb, at the very least, is 7.5x times faster than the existing efficient CL formulations.

types useful in recently proposed settings [40]. Since GDumb applies a mask (given a context) only at inference, we can dynamically adapt to the context to generate a mask. Similarly, we can extend GDumb beyond deterministic oracles ($m_i \in \{0, 1\}$) to probabilistic one ($m_i \in [0, 1]$). This delivers a lot of flexibility for diverse extensions like cost-sensitive classification, class-imbalance among others.

5 Conclusion

In this work, we provided a general view of a continual image classification problem. We then proposed a simple and general approach with minimal restrictions and empirically showed that it outperforms almost all the complicated state-of-the-art approaches *in their own formulations* for which they were specifically designed. We hope that our approach serves the purpose of a strong baseline to benchmark the effectiveness of any newly proposed CL algorithm. Our solution also raises various concerns to be investigated: (1) Even though there are plenty of research articles focused on specific scenarios relating CL problem, are we really progressing in the right direction? (2) Which formulation to focus on? and (3) Do we need different experimental formulations, more complex than the current ones, so that the effectiveness of recent CL models, if they are, is pronounced?

Acknowledgements: AP would like to thank Aditya Bharti, Shyamgopal Karthik, Saujas Vaduguru, and Aurobindo Munagala for helpful discussions. PHS and PD thank EPSRC/MURI grant EP/N019474/1, and Facebook (DeepFakes grant) for their support. This project was supported by the Royal Academy of Engineering under the Research Chair and Senior Research Fellowships scheme. PHS and PD also acknowledge FiveAI UK.

References

1. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation. (1989)
2. Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A., Bengio, Y.: An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211 (2013)
3. Li, Z., Hoiem, D.: Learning without forgetting. TPAMI (2017)
4. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: CVPR. (2017)
5. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: ICML. (2017)
6. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. In: PNAS. (2017)
7. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. In: NeurIPS. (2017)
8. Chaudhry, A., Dokania, P.K., Ajanthan, T., Torr, P.H.: Riemannian walk for incremental learning: Understanding forgetting and intransigence. In: ECCV. (2018)
9. De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., Tuytelaars, T.: Continual learning: A comparative study on how to defy forgetting in classification tasks. arXiv preprint arXiv:1909.08383 (2019)
10. Scheirer, W., Rocha, A., Sapkota, A., Boult, T.: Towards open set recognition. TPAMI (2012)
11. Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Charlin, L., Tuytelaars, T.: Online continual learning with maximally interfered retrieval. In: NeurIPS. (2019)
12. Jin, X., Du, J., Ren, X.: Gradient based memory editing for task-free continual learning. (2020)
13. Dhar, P., Vikram Singh, R., Peng, K.C., Wu, Z., Chellappa, R.: Learning without memorizing. In: CVPR. (2019)
14. Zhang, J., Zhang, J., Ghosh, S., Li, D., Tasci, S., Heck, L., Zhang, H., Kuo, C.C.J.: Class-incremental learning via deep model consolidation. In: WACV. (2020)
15. Yu, L., Twardowski, B., Liu, X., Herranz, L., Wang, K., Cheng, Y., Jui, S., Weijer, J.v.d.: Semantic drift compensation for class-incremental learning. In: CVPR. (2020)
16. Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., Fu, Y.: Large scale incremental learning. In: CVPR. (2019)
17. Hou, S., Pan, X., Loy, C.C., Wang, Z., Lin, D.: Learning a unified classifier incrementally via rebalancing. In: CVPR. (2019)
18. Castro, F.M., Marín-Jiménez, M.J., Guil, N., Schmid, C., Alahari, K.: End-to-end incremental learning. In: ECCV. (2018)
19. Belouadah, E., Popescu, A.: Il2m: Class incremental learning with dual memory. In: ICCV. (2019)
20. Zhao, B., Xiao, X., Gan, G., Zhang, B., Xia, S.T.: Maintaining discrimination and fairness in class incremental learning. In: CVPR. (2020)
21. Douillard, A., Cord, M., Ollion, C., Robert, T., Valle, E.: Small-task incremental learning. In: ECCV. (2020)
22. Liu, Y., Su, Y., Liu, A.A., Schiele, B., Sun, Q.: Mnemonics training: Multi-class incremental learning without forgetting. In: CVPR. (2020)

23. Rajasegaran, J., Hayat, M., Khan, S., Khan, F.S., Shao, L.: Random path selection for incremental learning. In: NeurIPS. (2019)
24. Rajasegaran, J., Khan, S., Hayat, M., Khan, F.S., Shah, M.: itaml: An incremental task-agnostic meta-learning approach. In: CVPR. (2020)
25. Abati, D., Tomczak, J., Blankevoort, T., Calderara, S., Cucchiara, R., Bejnordi, B.E.: Conditional channel gated networks for task-aware continual learning. In: CVPR. (2020)
26. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016)
27. Yoon, J., Lee, J., Yang, E., Hwang, S.J.: Lifelong learning with dynamically expandable network. In: ICLR. (2018)
28. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: NeurIPS. (2017)
29. Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y.W., Pascanu, R., Hadsell, R.: Progress & compress: A scalable framework for continual learning. In: ICML. (2018)
30. Yoon, J., Kim, S., Yang, E., Hwang, S.J.: Scalable and order-robust continual learning with additive parameter decomposition. In: ICLR. (2020)
31. Nguyen, C.V., Li, Y., Bui, T.D., Turner, R.E.: Variational continual learning. In: ICLR. (2018)
32. Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: ECCV. (2018)
33. Lee, S.W., Kim, J.H., Jun, J., Ha, J.W., Zhang, B.T.: Overcoming catastrophic forgetting by incremental moment matching. In: NeurIPS. (2017)
34. Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P.K., Torr, P.H., Ranzato, M.: Continual learning with tiny episodic memories. In: ICML-W. (2019)
35. Chaudhry, A., Gordo, A., Lopez-Paz, D., Dokania, P.K., Torr, P.: Using hindsight to anchor past knowledge in continual learning (2020)
36. Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with a-gem. In: ICLR. (2019)
37. Aljundi, R., Lin, M., Goujaud, B., Bengio, Y.: Gradient based sample selection for online continual learning. In: NeurIPS. (2019)
38. Tulving, E.: Episodic memory: From mind to brain. Annual review of psychology (2002)
39. Norman, K.A., O'Reilly, R.C.: Modeling hippocampal and neocortical contributions to recognition memory: a complementary-learning-systems approach. Psychological review (2003)
40. Ren, M., Iuzzolino, M.L., Mozer, M.C., Zemel, R.S.: Wandering within a world: Online contextualized few-shot learning. arXiv preprint arXiv:2007.04546 (2020)
41. Ji, X., Henriques, J., Tuytelaars, T., Vedaldi, A.: Automatic recall machines: Internal replay, continual learning and the brain. arXiv preprint arXiv:2006.12323 (2020)
42. Hsu, Y.C., Liu, Y.C., Kira, Z.: Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In: NeurIPS-W. (2018)
43. Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., Tesauro, G.: Learning to learn without forgetting by maximizing transfer and minimizing interference. In: ICLR. (2019)
44. Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T.P., Wayne, G.: Experience replay for continual learning. In: NeurIPS. (2019)

45. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: ICLR. (2017)
46. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: Regularization strategy to train strong classifiers with localizable features. In: ICCV. (2019)
47. Yin, H., Molchanov, P., Alvarez, J.M., Li, Z., Mallya, A., Hoiem, D., Jha, N.K., Kautz, J.: Dreaming to distill: Data-free knowledge transfer via deepinversion. In: CVPR. (2020)
48. Zeno, C., Golan, I., Hoffer, E., Soudry, D.: Task agnostic continual learning using online variational bayes. arXiv preprint arXiv:1803.10123 (2018)
49. Hocquet, G., Bichler, O., Querlioz, D.: Ova-inn: Continual learning with invertible neural networks. In: IJCNN. (2020)
50. van de Ven, G.M., Tolias, A.S.: Generative replay with feedback connections as a general strategy for continual learning. arXiv preprint arXiv:1809.10635 (2018)
51. Serra, J., Suris, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: ICML. (2018)
52. Rannen, A., Aljundi, R., Blaschko, M.B., Tuytelaars, T.: Encoder based lifelong learning. In: CVPR. (2017)
53. Mallya, A., Lazebnik, S.: Packnet: Adding multiple tasks to a single network by iterative pruning. In: CVPR. (2018)
54. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR. (2017)

GDumb: Appendix

Form.	Dataset	Model	#Params	k	Acc
B1	MNIST	MLP-400	479K	4400	97.8
		NIN	260K	250	97.2
		Improv	1.8x	17.6x	-0.6%
B1	SVHN	ResNet18	11.2M	4400	93.4
		NIN	968K	2000	89.3
		Improv	11.6x	2.2x	-4.1%
A1/A2	CIFAR10	ResNet18	11.2M	500	45.8
		DenseNet100	769K	500	47.5
		Improv	14.6x	-	1.7%
D	CIFAR100-Task	ResNet18-S	1108K	1105	60.3
		DenseNet100	794K	1105	62.1
		Improv	1.4x	-	1.8%
B2	CIFAR100-Class	ResNet18	11.2M	2000	24.1
		DenseNet100	800K	2000	27.5
		Improv	14.6x	-	3.4%

Table 1. Minimizing the resource consumption (storage size and parameters).

A Additional Experiments

We perform additional experiments to study variance of performance across memory sizes, models, training timesteps on each dataset. We first study the variation across models and different values of k and tabulate these results in Table 2. For MNIST and SVHN, we could reduce the replay buffer size by over 17x and 2x respectively with small impact on performance, a good tradeoff. For CIFAR10 and CIFAR100, we substitute the bulky ResNet18 model with a DenseNetBC-100-12 [54] which gives us a major decrease of over 14x in parameters. In CIFAR10 and CIFAR100 (Task-IL and Class-IL), we additionally improve accuracy by 1.7%, 1.8% and 3.4% respectively after compressing the models. On MNIST and SVHN, we could compress the size of stored dataset k , showing existing formulations use too much memory. But as dataset complexity increases, larger memory size is required. We could also compress the parameter storage by large margins while increasing accuracy by substituting popular but efficient models, showing the scope for using more efficient models for CL formulations.

We further study the trade-off between accuracy and training time of our improved models on the improved k values by varying the passes on memory.

Passes/Form.	B1-MNIST	B1-SVHN	A1-CIFAR10	D	B2	C2
k	250	2000	500	1105	2000	9000
	NIN	NIN	DenseNet100	DenseNet100	DenseNet100	DenseNet100
8	91.7 (96.0)	72.5 (81.4)	28.4 (28.4)	49.5 (50.8)	7.3 (8.8)	32.3 (33.9)
16	95.9 (96.3)	85.1 (85.6)	32.3 (33.5)	52.6 (53.0)	10.0 (11.4)	39.4 (41.9)
32	96.9 (**97.2**)	88.8 (87.3)	37.8 (36.7)	56.2 (58.3)	15.0 (18.1)	47.2 (48.4)
64	97.4 (97.2)	**89.2** (87.5)	39.9 (40.1)	60.6 (61.2)	21.9 (22.5)	54.0 (54.3)
128	97.4 (97.2)	89.2 (88.9)	43.9 (42.9)	**62.1** (61.9)	26.5 (24.1)	**56.9** (56.3)
256	96.6 (97.5)	88.4 (89.7)	46.4 (43.9)	62.0 (62.4)	**27.6** (25.7)	54.1 (54.8)
512	96.1 (97.7)	86.9 (88.4)	**47.5** (45.9)	62.3 (61.5)	27.6 (25.8)	54.0 (54.7)

Table 2. Accuracy of tweaked (NIN/DenseNet) GDumb models with number of passes. The bolded accuracies represent the reported results in previous experiments, while accuracies in (brackets) are obtained without cutmix regularization. We can halve the training time with slight tradeoff of upto 1% accuracy.

We present the results in Table 2, with **bold** being the selected models, along with (brackets) representing accuracies obtained by an ablation without cutmix regularization. We show that we can further reduce our training time by half with a minor (approx 1%) tradeoff in accuracy. We also observe that cutmix regularization improves performance by 0.2% to 1.5% margins. We strongly recommend regularization in GDumb since it only has access to the few samples in memory.

B Experiment formulations

We first detail each of the seven selected formulation and then present results on each of them.

Formulation A1 ([11]): We benchmark on two datasets: MNIST and CIFAR10 from this setting. They randomly divided MNIST and CIFAR10 into 5 disjoint tasks of 2 classes each. The architectures used for MNIST is a MLP with 2 hidden layers of 400 nodes and ResNet18 for CIFAR10. They use a small limit (k) of 300 and 500 stored samples for MNIST and 200, 500 and 1000 stored samples for CIFAR10. In MNIST, they select 1000 samples per task, while in CIFAR10 they utilize 9,750 samples per task in the online stream. On MNIST, we additionally compare with GSS [37] who use the same setup except they have an MLP with 100 hidden size. We compare accuracy on the hold-out test set after all tasks are done.

Formulation A2 ([12]): They split MNIST and CIFAR10 into 5 disjoint subsets by their labels as different tasks. Each task consists of 1,000 online training examples in MNIST and 10,000 training examples in CIFAR10 similar to the above setup. The goal is to classify over all 10 classes on a held-out test set when the training ends. The architectures used for MNIST is a MLP with 2 hidden layers of 100 nodes and ResNet18 for CIFAR10, identical to the above

setup. However, since the accuracies obtained for ER, GEM and ER-MIR are very different, we list them as a different formulation and compare our performance against them.

Formulation A3 ([41]): They split MNIST and CIFAR10 into 5 disjoint subsets. Each task consists of 1,000 online training examples in MNIST and 9,750 training examples in CIFAR10 similar to the A1. The goal is to classify over all 10 classes on a held-out test set when the training ends. The architectures used for MNIST is a MLP with 2 hidden layers of 100 nodes and ResNet18 for CIFAR10, identical to the above setup. However, since the accuracies obtained for ER, GEM and ER-MIR are very different (for MNIST), we list them as a different formulation as merging tables is not possible.

Formulation B1 ([42,23]): It consists of two datasets: MNIST and SVHN, randomly divided into 5 disjoint tasks of 2 classes each. The architecture used is a MLP with 2 hidden layers of 400 nodes for MNIST and ResNet18 for SVHN. The formulation controls the total static memory overhead among all proposed approaches, resulting in storage capacity of 4400 for memory-based approaches [42].

Formulation B2 ([4]): The formulation use CIFAR100, split into 20 tasks of 5 classes each. They use a ResNet-32 model and a limit of 2000 stored samples (k). We additionally report the average of accuracy after each task as described in [4] referred to as accuracy (avg in t), along with accuracy after all tasks referred to as accuracy.

Formulation B3 ([21]): This formulation tests small-task increments on CIFAR100 and Imagenet100. Given a class-order, they use the first 50 tasks for pretraining and then subsequent 50 tasks with 1 class each in a CI-CL fashion including the initial 50 classes. Hence, they have a CI-CL classification with 51-100 classes. They start off with 1000 samples in memory and add 20 samples to memory for each subsequent task added. They use ResNet32 on CIFAR100 and ResNet18 on Imagenet100 to match the formulation and provide 3 class-orders for CIFAR100 and 1 class-order for Imagenet100 which we use. We measure average and last-task accuracy on using the same set of class-orders, using the same memory and networks as specified in the formulation.

Formulation C1 ([42]): It splits MNIST into 5 disjoint tasks of 2 classes each. The architecture used is a MLP with 2 hidden layers of 400 nodes for MNIST. The formulation controls the total static memory overhead, resulting in storage capacity of 4400 for memory-based approaches. We use three different class-to-task mappings to get performance.

Formulation C2 ([9]): TinyImagenet is divided into 10 disjoint tasks of 20 classes each in this formulation. We compare with the overall best accuracies obtained (Table 10 in [9]). The table lists their best performance observed over different architectures, regularization strategies, hyperparameter searches, etc. We test for two stored sample limit (k): 4500 and 9000. We use DenseNetBC-100-12 architecture as detailed in subsequent sections. Note that although it differs from the architectures tested; it is a fairly standard efficient architecture.

Formulation D ([36]): We borrow the benchmark on CIFAR100, consisting of 20 disjoint tasks of 5 classes each. We train 17 tasks, store upto 13 samples per class and use the same reduced ResNet18 architecture.

Formulation E ([37]): It consists of two datasets (MNIST and CIFAR10) with class-imbalanced, blurry boundary setting. Each has 5 tasks, each task having 2 classes each. MNIST has 2000 online samples of current task and 200 from each other tasks for every task in the formulation. Similarly, in CIFAR10 we keep 90% of the data for each task, and introduce 10% of data from the other tasks. The same architectures are used as in GSS [37], which are a 2-layer MLP with hidden size 100 for MNIST and a ResNet18 for CIFAR10. The limit on stored samples (k) is 300 for MNIST and 500 for CIFAR10.