Knowledge Capture and Replay for Continual Learning

Saisubramaniam Gopalakrishnan 1,4, Pranshu Ranjan Singh 1,4, Haytham Fayek 3, Savitha Ramasamy 1,2,4, ArulMurugan Ambikapathi 1,2,4
1 Institute for Infocomm Research (I2R), A*STAR, Singapore *
2 Artificial Intelligence, Analytics And Informatics (A13), A*STAR, Singapore
3 RMIT University, Australia
4 CNRS@CREATE LTD, 1 Create Way, 08-01 CREATE Tower, Singapore 138602 †

Abstract

Deep neural networks model data for a task or a sequence of tasks, where the knowledge extracted from the data is encoded in the parameters and representations of the network. Extraction and utilization of these representations is vital when data is no longer available in the future, especially in a continual learning scenario. We introduce flashcards, which are visual representations that capture the encoded knowledge of a network as a recursive function of some predefined random image patterns. In a continual learning scenario, flashcards help to prevent catastrophic forgetting by consolidating the knowledge of all the previous tasks. Flashcards are required to be constructed only before learning the subsequent task, hence, they are independent of the number of tasks trained before, making them task agnostic. We demonstrate the efficacy of flashcards in capturing learned knowledge representation (as an alternative to the original data), and empirically validate on a variety of continual learning tasks: reconstruction, denoising, and task-incremental classification, using several heterogeneous (varying background and complexity) benchmark datasets. Experimental evidence indicates that: (i) flashcards as a replay strategy is task agnostic, (ii) performs better than generative replay, and (iii) is on par with episodic replay without additional memory overhead.

1. Introduction

Despite the success of deep neural networks in numerous domains [5, 11, 15], it remains a challenge to extract and reuse the knowledge embedded in the representations of trained models for similar or other downstream scenarios [2]. Approaches such as transfer learning [25] and knowledge distillation [7] enable representational or functional translation [9] from one model to the other. However, they cannot maintain a unified representation that evolves with the characteristics of data. Knowledge captured from the past should be consolidated and preserved as some representation with low computational and memory expense, especially, when the past data is no longer available.

In particular, learning new tasks affect retention of previous knowledge [19]. Continual learning (CL) [21] was
proposed as a candidate for such retention, and is gaining attention [18]. Essentially, CL aims to make deep neural networks learn continually from a sequence of tasks, without catastrophically forgetting the past sequences/tasks. There is a need to develop a reliable method that can effectively capture and re-purpose knowledge from a learned model, which can then be exploited in the CL setting. While CL approaches are predominantly studied on classification (class/task incremental learning), its potential remains relatively less explored in other scenarios such as task agnostic unsupervised reconstruction, denoising etc.

In this work, we introduce flashcards for capturing consolidated learned knowledge representations of all the previous tasks. A schematic illustration is shown in Figure 1. Random image patterns, when passed recursively through a trained autoencoder (obtained at end of step (a)), capture its representations with each pass, and are transformed into flashcards (step (b)). Step (c) uses the obtained flashcards to train a new network (single task), and step (d) addresses the multi-task / continual setting. We first evaluate flashcards’ potential for capturing representations of the data (as seen by the network) and later extend their usefulness for replay while learning continually from several heterogeneous datasets, such as MNIST, Fashion MNIST, Cifar10, SVHN, Omniglot, ImageNet, and UCMerced for a variety of applications such as reconstruction, denoising, and task-incremental classification. To the best of our knowledge, we are the first to report continual reconstruction and de-incremental classification. To the best of our knowledge, SVHN, Omniglot, ImageNet, and UCMerced for a variety of datasets, such as MNIST, Fashion MNIST, Cifar10, for replay while learning continually from several heterogeneous data (as seen by the network) and later extend their usefulness for capturing representations of the data (as seen by the network).

2. Knowledge Capture and Replay

We first introduce the notion of knowledge capture using flashcards and describe their construction from a trained autoencoder 1. Consider training an autoencoder for reconstruction task $T_t$ using dataset $D_t$ containing $N_t$ samples, where $D_t = \{X_1, X_2, \ldots, X_{N_t}\} \subset \mathbb{R}^{k \times l \times c}$, is the set of training image samples with $k$ rows, $l$ columns, and $c$ channels. For the task $T_t$, the autoencoder is trained to maximize the likelihood $P(X|\theta_t), \forall X \in D_t$ using the conventional mean absolute error (MAE; Eqn. (1)) between the original and reconstructed images:

$$\min_{\theta_t} \frac{1}{N_t} \sum_{n=1}^{N_t} |X_n - \hat{X}_n|, \tag{1}$$

where $\theta_t$ is the autoencoder network parameters (weights, biases, and batch norm parameters) for task $T_t$, $X_n$ is a sample from the empirical data distribution of $P(X)$, and $\hat{X}_n = f_t(X_n, \theta_t)$ is the reconstructed sample. $|X_n - \hat{X}_n|$ denotes the pixel-wise mean absolute error between $X_n$ and $\hat{X}_n$, and $f_t(\cdot)$ is the function approximated by the autoencoder to learn (reconstruct) task $T_t$. In the above conventional learning setup, the parameters $\theta_t$ of an autoencoder network aim to model the knowledge in the data $D_t$, such as shape, texture, color of the images. Let the reconstruction error of the trained autoencoder be bounded by $[\epsilon_1, \epsilon_2]$, i.e.,

$$\epsilon_1 \leq |X_n - \hat{X}_n| \leq \epsilon_2, \forall X_n \in D_t, X_n \sim P(X) \tag{2}$$

Let $P(M)$ be a different, but well-defined distribution in the same dimensional space as $P(X)$ (i.e., $\mathbb{R}^{k \times l \times c}$), and $D_m = \{M_1, M_2, \ldots, M_{N_f}\}$, where $M_i \in \mathbb{R}^{k \times l \times c}, M_i \sim P(M), i = 1, \ldots, N_f$. The output of the autoencoder for any $M_i \in D_m$ is

$$\hat{M}_i = f_t(M_i, \theta_t). \tag{3}$$

Since the autoencoder is trained for task $T_t$ and as $M_i \in D_m$ is sampled from another distribution, $\hat{M}_i$ will be a meaningless reconstruction of $M_i$. Alternatively, it can be said that the activations obtained on passing $M_i \sim P(M)$ through the trained $f_t(\cdot, \theta_t)$ does not align with $T_t$. This can be observed from the increased MAE between $M_i$ and $\hat{M}_i$, and from the t-SNE representations of the bottleneck layer (latent space) for several samples drawn from $P(M)$ (first two iterations of Figure 2).

Let $F_{r1}$ be the output after a series of $r$ recursive iterations/passes of $M_i$ through the trained autoencoder ($f_t(\cdot, \theta_t)$), i.e.,

$$F_{r1} = f_t^r(f_t^{r-1}(\ldots f_t(\{M_i, \theta_t\}) \ldots)). \tag{4}$$

Such recursive passing (output $\rightarrow$ input $\rightarrow$ output) gradually attunes the input $M_i$ towards $F_{r1}$ to produce activations that are accustomed to $\theta_t$. This can be observed in Figure 2 where the MAE reduces over recursive iterations, and the latent space representations of samples drawn from $P(M)$ begin to overlap with those from $P(X)$. Let

$$\gamma_1 \leq |F_{r1} - F_{(r-1)i}| \leq \gamma_2, \forall M_i \in D_m, M_i \sim P(M), \tag{5}$$

where $\gamma_1$, and $\gamma_2$ are the lower and upper bounds of reconstruction error for any $M_i \in D_m$. The aim now is to have a collection set $F_t = \{F_{r1}\}_{i=1}^{N_f}$ for task $T_t$ such that the following properties are satisfied:

- P1: $\gamma_2 < \epsilon_1$ for $\forall M_i \in D_m$,
- P2: Respective latent space distribution of $F_t \sim D_t$.

Such a set of constructions ($F_t$) obtained through recursive iterations of random inputs (drawn from $P(M)$) are defined

\begin{itemize}
  \item \textbf{P1:} $\epsilon_1 \leq |X_n - \hat{X}_n| \leq \epsilon_2, \forall X_n \in D_t, X_n \sim P(X)$
  \item \textbf{P2:} Respective latent space distribution of $F_t \sim D_t$.
\end{itemize}
Figure 2. A sample of three Flashcards ($F_t$) constructions over recursive iterations, with Cifar10 as an example, difference in MAE between successive iterations with iteration number and t-SNE of latent space in 2D. It can be observed that the initial (raw) input images adapt to the texture captured by the model, as the number of recursive iterations increase (iteration $t = 10, 20$). Also, the corresponding latent space clusters get closer. However, on further repeated passes ($r \geq 50$), the reconstruction deteriorates / smooth out (as MAE keeps reducing) and the two clusters drift apart again (P1 is satisfied but P2 is not). Best viewed in color.

Figure 3. Transformation from maze patterns ($D_m$) into Flashcards ($F_t$) for tasks: MNIST, Fashion MNIST and Cifar10.

as flashcards. These flashcards are expected to capture the knowledge from the trained network and serve as a potential alternative to $D_t$. Figure 3 illustrates a few flashcards constructed with same set of random images, from AEs trained with different datasets. Although these flashcards are intended to capture the knowledge representations in $\theta_t$ as a function of $M_i \in D_m$, it can be observed from Figure 3 that they do not bear direct shape similarities with their respective datasets. In fact, during the recursive process, the raw input image patterns are modified/transformed to textures that are suited to the trained autoencoder model (colored textures in flashcards are observed for Cifar10). This is in-line with [6] which shows that neural networks learn predominantly texture than shape by default. Now, since an autoencoder is trained to maximize $\log P(X|\theta_t)$, the following fact holds:

**Fact 1:** Let $D_A$ and $D_B$ be two i.i.d. datasets whose elements are drawn from $P(X)$. If there exists a trained autoencoder $f_A(\cdot; \theta_A)$, to reconstruct $D_A$ with error $[\beta_1, \beta_2]$, $\forall X \in D_A$, then the autoencoder $f_A(\cdot; \theta_A)$ can reconstruct $D_B$ with the same error range.

The converse is not true because any image dataset $D_B$ with images that have few pixels with non-zero values (other pixels as zeros), can still yield smaller reconstruction error and not necessary to be drawn from $P(X)$ (satisfying P1). Hence, both properties P1 and P2 for $F_t$ need to be satisfied. As P1 and P2 are dependent on the initial input distribution $P(\mathbf{M})$, it is important to have a suitable $P(\mathbf{M})$ to get $F_t$ ($P(\mathbf{M}) = P(X)$ is one trivial and uninteresting solution). As finding a perfect $P(\mathbf{M})$ is an ideal research problem by itself, in this work, we use random maze pattern images as a potential candidate $D_m$ to get $F_t$. The above leads to the following hypothesis:

**Hypothesis 1:** Since the autoencoder trained on $D_t$ also reconstructs elements from $F_t$ (Property P1) and has matching latent space distribution (Property P2), flashcards ($F_t$) can be used as a set of pseudo-samples for $D_t$ to learn a new autoencoder $f_i(\cdot; \theta_{\text{new}})$, when trained from scratch using only $F_t$.

### 2.1. Flashcard Construction

**Algorithm 1 Flashcard construction**

Require: $f_i(\cdot; \theta_t)$: Autoencoder model for task $T_i$

Require: $N_f$: Number of flashcards

Require: $r$: Number of recursive iterations through $f_k(\cdot; \theta_t)$

1: Let $D_m \leftarrow \{M_1, \ldots, M_{N_f}\}$, where $M_i \sim P(\mathbf{M}), \forall i$

2: $F_i \leftarrow \{\}$, where $F_i$ denotes the set of flashcards

3: for $i \in \{1, \ldots, N_f\}$

4: $F_{ri} = f_i(f_i^{-1}(\cdots f_i^{-1}(M_i, \theta_t) \cdots))$

5: $F_{t} = F_t \cup \{F_{ri}\}$

6: end for

7: return $F_t$

Algorithm 1 provides the steps to construct flashcards using a trained autoencoder model. For flashcards construction, the following hyperparameters are useful - (i) number of recursive iterations ($r$), (ii) number of flashcards to construct ($N_f$), (iii) choice of initial input distribution ($P(\mathbf{M})$). Figure 4 portrays empirical analysis of selecting the optimal...
iteration, considering two datasets - SVHN and Cifar10. Figure 4 (columns 1,3) shows Frechet Latent Space Distance (FLSD) is inspired from FID [8] and calculated as the Frechet distance between latent space activations of flashcards versus original samples (smaller the better), from the eyes of the trained autoencoder) (Orange) and MAE (Blue), as a function of $r$. Initially, increase in $r$ leads to decrease in FLSD, up to certain iterations, this indicates that flashcards distribution is getting closer to learned distribution. As $r$ increases further, though MAE keeps going down, FLSD increases, because propagation of reconstruction error in autoencoder causes drift in the features. Based on the empirical analysis $10 \leq r \leq 20$ is a sensible choice that worked well across all of our experiments. Also, reconstruction error (Figure 4, columns 2,4) is observed to be low in this iteration range. It further indicates that the flashcards are not critically sensitive to $r$. In all experiments $r$ is set to 10.

Finding the optimal $P(M)$ is a research question in itself. For our experiments, the choice of $P(M)$ is maze-like random image patterns since there is resemblance to edges and shapes. Gaussian random noise was also considered but there was lack of diversity among the constructed flashcards. Using an additional dataset (such as next task images) did not help at all. A comparative study among the different choices is provided at the end of Section 4.1.

Generally, performance of a task is dependent on the number of flashcards constructed with sufficient diversity, and having more flashcards indicate better performance, similar to the trend with coreset exemplars. Note from [17], that memory requirement for experiments is calculated based on network parameters and is independent of generated samples.

### 2.2. Flashcards for Capturing Representations

We first verify Hypothesis 1 by demonstrating the efficacy of flashcards to capture representations for single-task. **Alternative to Original Dataset:** Consider three different datasets: MNIST, Fashion MNIST, and Cifar10. Let 3 different autoencoders (Network1), all with same architecture, be trained for each of these 3 datasets separately using Eqn. (1) as loss function. Flashcards ($F_i$) constructed from each of these (Network1) autoencoders, for the same set of random input maze pattern images ($D_m$) (few samples shown in Figure 3) are used respectively to train new autoencoder (Network2) models from scratch (with the same architecture). Results in Table 1 show reconstructions are very close to the autoencoders trained on originals. Flashcards also work for large-scale resolution, as seen from Figure 5. An autoencoder is trained on ImageNet 256x256px resolution (1M images) and has a reconstruction error of 0.10. A separate autoencoder is trained using flashcards only of varying percentages. The error when the network is randomly initialized without any data is high (0.42), but with increase in number of flashcards, it drops between 0.17 to 0.13, matching closely with the original benchmark.

Results from Table 1 and Figure 5 confirm the hypothesis that flashcards indeed capture network parameters as a function of $D_m$, therefore be used as pseudo-samples / training data for both small and large-scale resolutions (as network parameters involved in flashcard construction are initially learned by training with the original dataset $D_o$).

**Data-Free Knowledge Distillation:** Consider training an autoencoder teacher network AE1 with data $D_{T_1}$. Once

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Network 1’s MAE</th>
<th>Network 2’s MAE</th>
<th>Alpha Original</th>
<th>Alpha Flashcards</th>
<th>Alpha Un-trained</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>0.0184</td>
<td>0.0491</td>
<td>1.9185</td>
<td>2.0531</td>
<td>1.4343</td>
</tr>
<tr>
<td>Fashion</td>
<td>0.0259</td>
<td>0.0440</td>
<td>2.0044</td>
<td>2.1158</td>
<td>1.4343</td>
</tr>
<tr>
<td>Cifar10</td>
<td>0.0564</td>
<td>0.0686</td>
<td>2.0266</td>
<td>2.1531</td>
<td>1.4343</td>
</tr>
</tbody>
</table>

Table 1. Reconstructions from two separate autoencoders, one trained on original data (Network1), and the other trained on flashcards (Network2). The metrics indicate that flashcards are sufficient alternatives towards learning the original data. Weighted Alpha [16] is based on HT-SR Theory. Alpha values are closer for the trained networks which indicates similarity between the two network weights. Reported values are averaged over 5 runs.
Figure 5. Reconstruction error of two networks, one trained on original ImageNet 256x256 resolution 1M images (grey), and the second trained only on flashcards 256x256 resolution (orange). X-axis denotes the % of samples added, and as more flashcards are introduced, error of the second network approaches closer to first.

Table 2. Building autoencoder AE2 (smaller and larger arch.) using flashcards obtained from AE1 (Variant I and II, each with different parameters) trained on Cifar10 dataset. Column Test Original refers to test MAE on original ImageNet 256x256 resolution 1M images (grey), and the respective strategies. Regularization strategies such as [10, 24] learn new tasks while imposing constraints on the network parameters to avoid deviating too much from the characteristics of both regularization (refer Algorithm 2 point 4) and replay, and will therefore be compared with the respective strategies. Regularization strategies such as [10, 24] learn new tasks while imposing constraints on the network parameters to avoid deviating too much from the characteristics of both regularization (refer Algorithm 2 point 4) and replay, and will therefore be compared with the respective strategies.

<table>
<thead>
<tr>
<th>Train Arch. Type</th>
<th>Params</th>
<th>Test Original</th>
<th>Test AE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE1 - Variant I</td>
<td>94,243</td>
<td>0.0640</td>
<td>–</td>
</tr>
<tr>
<td>AE2 - Smaller arch.</td>
<td>24,083</td>
<td>0.0787</td>
<td>0.0963</td>
</tr>
<tr>
<td>AE2 - Larger arch.</td>
<td>372,803</td>
<td>0.0512</td>
<td>0.0570</td>
</tr>
<tr>
<td>AE1 - Variant II</td>
<td>298,947</td>
<td>0.0437</td>
<td>–</td>
</tr>
<tr>
<td>AE2 - Smaller arch.</td>
<td>24,083</td>
<td>0.0358</td>
<td>0.0389</td>
</tr>
<tr>
<td>AE2 - Larger arch.</td>
<td>372,803</td>
<td>0.0512</td>
<td>0.0570</td>
</tr>
</tbody>
</table>

3. Related Works

In general, continual learning algorithms are based on architectural strategies, regularization, memory replay, and their combinations [18]. By the intrinsic nature of flashcards (as discussed in Sections 2.2 and 2.3), they implicitly exhibit the characteristics of both regularization (refer Algorithm 2 point 4) and replay, and will therefore be compared with the respective strategies. Regularization strategies such as [10, 24] learn new tasks while imposing constraints on the network parameters to avoid deviating too much from the characteristics of both regularization (refer Algorithm 2 point 4) and replay, and will therefore be compared with the respective strategies.
much from those learned from the previous tasks. [13] uses current task samples to regularize the soft labels of past tasks, thus avoiding explicit data storage. However, it is important to note that these regularization methods perform well in homogeneous (sharing similar characteristics) task environments, where it is possible to find mutually sub-optimal points in the solution space. However, such methods suffer when subsequent tasks are from different domains / heterogeneous datasets [13].

In a rehearsal mechanism, subset of samples from previous tasks, referred to as coreset samples / exemplars, serve as memory replay [3, 4, 14]. However, they suffer from excessive storage requirements that grows with the number of tasks. On the other hand, generative replay approaches [1, 12, 22, 20] face challenges in scalability to complex datasets with many tasks, since they involve heavy training and computation associated with an auxiliary network to generate visually meaningful images for each task.

Approaches discussed above are either suitable only for homogeneous data, or are memory intensive as they involve preserving samples, or computationally intensive in “generating” samples, or require task identifiers. The proposed flashcards are constructed on the fly, requires only simple autoencoder, independent of the number of tasks, and helpful both as regularization and replay mechanism. Flashcards can be used as pseudo samples constructed with low computational and memory expense. Furthermore, flashcards can scale across tasks without considerable drop in performance. Table 3 lists overall comparison on various traits between flashcards, episodic, and generative replay.

### 4. Experiments in Continual Learning

We provide experimental validation of using flashcards in CL scenario for different applications such as: (i) continual reconstruction (Section 4.1), (ii) continual denoising (Section 4.2), and new task incremental classification (Section 4.3). For all experiments, we compare with upperbound (Joint Training, JT), lowerbound (Sequential Fine-tuning, SFT), and standard methods for regularization, generative and episodic replay as baseline. To the best of our knowledge, we are the first to report continual reconstruction and denoising. Our results indicate that using flashcards indeed helps under heterogeneous conditions spanning different domains, and where the task identifier is not available.

**Datasets:** For the continual reconstruction and denoising, 5 heterogeneous public datasets, referred as Sequence5 was employed. The datasets used in Sequence5 are in the order - MNIST, Fashion MNIST, Cifar10, SVHN, and Omniglot. Each dataset was considered as a task, with class labels omitted. MNIST, Fashion MNIST and Omniglot were resized to 32x32x3 (bilinear rescale and channel copied twice) to maintain the same scale as the other two datasets. Continual reconstruction is also compared on UC Merced Land Use [23], a real-world large-scale remote sensing dataset with 256x256 resolution and 21 classes. For task incremental classification, Sequence3 comprising of the order - Cifar10, MNIST and Fashion MNIST was employed.

**Architecture and Training Details:** For reconstruction and denoising, flashcards are constructed using an autoencoder architecture with 4 layers of down/upsampling and 64 convolutional filters per layer. The bottleneck dimension 256 offers 12x reduction from image space, serving to demonstrate the effectiveness of forgetting. The architecture is a simplified variant of the VGG, adapted with fewer layers and filters for proper reconstruction. All hidden layers employ tanh activation.

For task incremental classification, the architecture of classifier is based on specifications from [22]. For the architecture of autoencoder, the encoder weights are pre-initialized with classifier encoder weights, decoder is attached after latent space, and the autoencoder is trained by passing current task samples along with flashcards up to current task. For the next task, classifier uses new task samples along with constructed flashcards of previous task. Soft labels for flashcards are obtained by passing them through the classifier, before the start of next task.

**Training details:** For reconstruction and denoising,
network is trained for 100 epochs per task, and optimized using Adam and learning rate of 0.001. Each minibatch update is based on equal number of flashcards and the current task samples. 10% of the training data is allocated for validation. Early stopping is employed if there is no improvement for over 20 epochs. For task incremental classification, mini-batch size of 256 and Adam optimizer with learning rate=0.0001 is used. Both classifier and autoencoder models are trained for 5K iterations. 5K flashcards with $\lambda = 1$ (refer Algorithm 2) was employed for replay.

### 4.1. Heterogeneous Continual Reconstruction

Continual reconstruction is an experiment to study the ability of different continual learning methods to effectively remember the characteristics of data learned thus far to reconstruct samples of both past and present. A single network is trained on multiple heterogeneous datasets incrementally, and during inference, it is expected to remember the task of reconstruction without an explicit task identifier. We measure individual task MAE (averaged) and BWT, customized for reconstruction. Results for reconstruction using Sequence5 (details in Section 4) are available under columns 5 and 6 in Table 4. Since continual reconstruction for homogeneous cases is trivial and does not require extra effort (as observed from Table 5), we focus on the more challenging heterogeneous scenario where forgetting is inevitable. Comparison with baseline methods is summarized in the captions of Table 4. We compare flashcards against different continual learning based regularization and replay techniques and observe that it outperforms most baselines user fewer network capacity. It performs on-par with episodic replay methods without occupying external memory for past samples.

Continual reconstruction results over 21 tasks on UC Merced land use remote sensing dataset (256px) is illustrated in Figure 6. Comparison is made per task (on x-axis) against SFT, Coreset (max 100 samples in buffer), and VAE+AE, with JT as reference line. All autoencoders require only 2.9MB (for 256x256 dim) in terms of network parameters. Coreset requires extra 75MB for storing 100 images. VAE occupies an extra 4.7MB. Both Flashcards and VAE+AE generate 100 samples towards next task and have similar performance. VAE+AE is slightly better than Flashcards at the expense of more storage. Other methods experience spike in transition to next task, whereas the performance is stable when using flashcards.

### Table 4. Continual learning for Sequence5 reconstruction and denoising. We provide a comparison with different methods in terms of error observed MAE and BWT on test set. Additionally, we compare the Network Capacity (N/w (MB)) and External Memory (Mem (MB)) required by each method. Our method performs better than generative method based replay, and is on-par with the error on using episodic memory based coreset replay. Further, our memory requirement is very less, and an external memory buffer to store past tasks is not required. Lower Avg MAE (↓) and higher BWT (↑) are better. Reg=Regularization, Reg+R=Regularization and Replay, ER=Episodic Replay, GR=Generative Replay, FR=Flashcard Replay. Reported scores are averaged over 3 independent runs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>N/w Mem</th>
<th>Recon MAE ↓</th>
<th>Recon BWT ↑</th>
<th>Denoise MAE ↓</th>
<th>Denoise BWT ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>JT</td>
<td></td>
<td>1.5</td>
<td>798.7</td>
<td>0.0548 ± 0.000</td>
<td>0.0572 ± 0.000</td>
<td></td>
</tr>
<tr>
<td>SFT</td>
<td></td>
<td>1.5</td>
<td>5518 ± 0.001</td>
<td>−0.6385 ± 0.002</td>
<td>0.5477 ± 0.002</td>
<td>−0.6275 ± 0.002</td>
</tr>
<tr>
<td>Coreset 500</td>
<td>ER</td>
<td>1.5</td>
<td>0.0560 ± 0.005</td>
<td>−0.0246 ± 0.006</td>
<td>0.0748 ± 0.004</td>
<td>−0.0219 ± 0.005</td>
</tr>
<tr>
<td>Coreset 5000</td>
<td>ER</td>
<td>15.3</td>
<td>0.0495 ± 0.002</td>
<td>−0.0185 ± 0.002</td>
<td>0.0595 ± 0.002</td>
<td>−0.0172 ± 0.002</td>
</tr>
<tr>
<td>LwF</td>
<td>Reg+R</td>
<td>1.5</td>
<td>0.5495 ± 0.002</td>
<td>−0.5241 ± 0.002</td>
<td>0.4370 ± 0.002</td>
<td>−0.3338 ± 0.002</td>
</tr>
<tr>
<td>VAE 5000 + AE</td>
<td>GR</td>
<td>2.9</td>
<td>0.0751 ± 0.004</td>
<td>−0.0575 ± 0.006</td>
<td>0.0711 ± 0.005</td>
<td>−0.0438 ± 0.005</td>
</tr>
<tr>
<td>CL VAE 5000</td>
<td>GR</td>
<td>1.4</td>
<td>0.3524 ± 0.041</td>
<td>−0.1658 ± 0.038</td>
<td>0.5460 ± 0.042</td>
<td>−0.2839 ± 0.033</td>
</tr>
<tr>
<td>Flashcards 5000</td>
<td>FR</td>
<td>1.5</td>
<td>0.0536 ± 0.002</td>
<td>−0.0277 ± 0.002</td>
<td>0.0588 ± 0.002</td>
<td>−0.0262 ± 0.002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Cifar10</th>
<th>Caltech101</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_1$</td>
<td>$T_5$</td>
</tr>
<tr>
<td>JT</td>
<td>0.0596</td>
<td>0.0596</td>
</tr>
<tr>
<td>SFT</td>
<td>0.0872</td>
<td>0.0748</td>
</tr>
</tbody>
</table>

Table 5. Continual reconstruction for homogeneous tasks (each task $T_k$ specifies upto k classes from the same dataset have been observed) shows minimal / no forgetting, examples showcasing reconstruction error for Cifar10 and Caltech101. On the contrary there is strong indication that knowledge from past improves learning newer tasks (forward transfer).

### Figure 6. Continual reconstruction on UC Merced land use 256px dataset. Line plot shows reconstruction error averaged from first till current task, as new tasks are introduced. SFT and Coreset oscillate occasionally as new tasks are encountered. E.g. beach (blue texture) after agriculture and baseball ground (green), forest after series of buildings (grey), golfcourse after freeway, overpass after park etc. Despite minimal capacity among other methods, Flashcards has relatively stable transition (dotted horizontal line shows peak error) and low error for all tasks.
Flashcards initialization: A study on the choice of initialization is presented in Table 7 and Figure 7 using Sequence5 for continual reconstruction. Among the three choices - Maze, Gaussian Noise and next task data, only Maze based flashcards had diversity in terms of shape and texture, and performed best due to its ability to capture activations at edges. Using Gaussian noise as initialization resulted in repetition of same patterns. Flashcards from new task initialization being dependent on the random subset images’ coverage of old tasks’ activations led to inconsistent results.

<table>
<thead>
<tr>
<th>Maze Pattern</th>
<th>Gaussian Noise</th>
<th>Next Task data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0536 ± 0.002</td>
<td>0.0945 ± 0.003</td>
<td>0.3713 ± 0.275</td>
</tr>
</tbody>
</table>

Table 7. Maze patterns provide best reconstruction MAE among different initializations used for construction of flashcards. Experiments run on Sequence5 continual reconstruction.

4.3. Task (Agnostic) Incremental Classification

Task Incremental Learning (Task-IL) is a continual learning setting where new tasks, each comprising of multiple classes (≥ 2) are learned in incremental fashion. The network encoder is shared whereas there exists task-exclusive multihead output. Generally, classification is performed using the specific multihead, with the help of task identifier. We compare performance of flashcards for both cases, with and without task identifier. We follow the same setup as provided in [22] (details in Section 4). Results for Sequence3 (details in Section 4) from Table 6, show that baseline methods designed to make decisions only on seeing the task identifier during inference perform poorly in its absence. Our approach, on the other hand, is designed as task agnostic, relying only on the most recent checkpoint. It is competitive in the presence of task identifier, and in the absence, it outperforms baseline methods by a big margin.

5. Conclusion

We introduced flashcards that can capture knowledge representations of a trained autoencoder through recursive passing of random image patterns, and showed that it can be used as alternative to original data. We further demonstrated its efficacy as a task agnostic replay mechanism for various continual learning scenarios, such as reconstruction, denoising, and task incremental learning, with heterogeneous datasets. Flashcard replay outperforms generative replay and regularization methods, without additional memory and training, and also performs on par with episodic replay, without storing exemplars. The intrinsic nature of flashcards allows for data abstraction that can be exploited for potential data privacy applications. Generalization of flashcards to other domains will foster further research.
References


