Class Incremental Online Streaming Learning

Soumya Banerjee\textsuperscript{1}, Vinay Kumar Verma\textsuperscript{2}, Toufiq Parag\textsuperscript{3}, Maneesh Singh\textsuperscript{3}, Vinay P. Namboodiri\textsuperscript{1,4}
\textsuperscript{1}IIT Kanpur, India \textsuperscript{2}Duke University, USA \textsuperscript{3}Verisk Analytics, NJ, USA \textsuperscript{4}University of Bath, UK

\texttt{soumyab@cse.iitk.ac.in, vinaykumar.verma@duke.edu, toufiq.parag@verisk.com}
\texttt{maneesh.singh@verisk.com, vpn22@bath.ac.uk}

Abstract

A wide variety of methods have been developed to enable lifelong learning in conventional deep neural networks. However, to succeed, these methods require a ‘batch’ of samples to be available and visited multiple times during training. While this works well in a static setting, these methods continue to suffer in a more realistic situation where data arrives in online streaming manner. We empirically demonstrate that the performance of current approaches degrades if the input is obtained as a stream of data with the following restrictions: (i) each instance comes one at a time and can be seen only once, and (ii) the input data violates the i.i.d assumption, i.e., there can be a class-based correlation. We propose a novel approach (CIOSL) for the class-incremental learning in an online streaming setting to address these challenges. The proposed approach leverages implicit and explicit dual weight regularization and experience replay. The implicit regularization is leveraged via the knowledge distillation, while the explicit regularization incorporates a novel approach for parameter regularization by learning the joint distribution of the buffer replay and the current sample. Also, we propose an efficient online memory replay and replacement buffer strategy that significantly boosts the model’s performance. Extensive experiments and ablation on challenging datasets show the efficacy of the proposed method.

1. Introduction

In this paper, we aim to achieve true continual learning by solving an extreme and restrictive form of lifelong learning. Predominantly, the more popular and successful methods in continual learning operate in incremental batch learning (IBL) scenarios \cite{1,23,37,40,42}. In IBL, we assume that current task data is available in batches during training, and the learner visits them sequentially multiple times. However, these methods are ill-suited in a continuously changing environment, where the learner needs to quickly adapt to the newly available data without catastrophic forgetting \cite{12} in an online manner.

The ability to continually learn effectively from streaming data with no catastrophic forgetting is a challenge that has not received widespread attention \cite{14}. However, its utility is apparent, as it enables the practical deployment of autonomous AI agents. In the real world, an autonomous agent continuously encounters new examples of a novel or previously observed classes. To adapt to these new examples, it is often required to train the AI agent with these new instances immediately without disrupting its service. It is infeasible to wait and aggregate a ‘batch’ of new samples for learning on the new data. Moreover, the ability to learn online in a streaming setting is not only practical, but a step towards enabling true lifelong learning on embodied AI agents in a dynamic, non-stationary environment.

In this work, we are interested in learning continuously in online streaming setting, where a learner needs to learn from a single sample at a time, with no catastrophic forgetting, in a single-pass. The model can visit any part of data only once, and it can be evaluated any time without waiting for termination of training. In addition, we also aim to evaluate the model in class-incremental setting, such that at test time, we consider the label space over all the observed classes so far. This is in contrast to task-incremental learning methods such as VCL \cite{31}, which requires task labels to be specified during inference. While some recent works aim at online learning and consider class-incremental learning setting, these methods are not suitable in the streaming learning setting. For instance, GDumb \cite{33} suggests that storing data points with a greedy-sampler and retraining before inference outperforms existing approaches. However, one does visit data points multiple times during retraining, which violates the single-pass learning constraint. AGEM \cite{7}, another online learning approach, is based on projected-gradient-descent, also suffers severe forgetting in streaming setting. Finally, the recently proposed streaming learning approach, REMIND \cite{15} is limited in terms of requiring a significant amount of cached data. Our work addresses all these limitations in a principled manner with a limited amount of information. Figure \ref{fig:Diagram} compares the proposed method with...
the recent strong baselines. We observe that Our-model outperforms the baselines by a significant margin in all three different lifelong learning settings. It implies that a learning method in the most restrictive setting can be thought of as a universal method for all lifelong learning settings with the widest possible applicability.

We propose a novel method, ‘Class Incremental Online Streaming Learning’ (CIOSL), which enables lifelong learning in online streaming setting by leveraging tiny episodic memory replay and regularization. It regularises the model from two sources. One focuses on regularizing the model parameters explicitly in an online Bayesian framework, while the other regularises implicitly by enforcing the updated model to produce similar outputs for previous models on the past observed data. Our approach is jointly trained with buffer replay and current task samples by incorporating the likelihood of the replay and current samples. As a result, we do not need explicit finetuning with buffer samples. We propose a novel online loss-aware strategy for buffer replacement and coreset sample selection that significantly boosts the model’s performance. Our approach only requires a small subset of samples for memory replay, and we do not use full buffer replay like past approaches [14]. Our experimental results on four benchmark datasets demonstrate the effectiveness as well as superiority of the proposed method to circumvent catastrophic forgetting over state-of-the-art baselines, and the extensive ablations validate the components of our method.

Our contributions can be summarised as follows: (i) we propose a novel dual regularization framework (CIOSL), comprising an online Bayesian framework as well as a functional regularizer, to overcome catastrophic forgetting in challenging online streaming learning scenario, (ii) we propose a novel online loss-aware buffer replacement and sampling strategy which significantly boosts the model’s performance, (iii) we empirically show that selecting a subset of samples from memory and computing the joint likelihood with the current sample is highly efficient, and enough to avoid explicit finetuning, and (iv) we experimentally show that our method significantly outperforms the state-of-the-art baselines.

Figure 1. $Ω_{\Omega}$ results for ‘batch’, ‘online’ and ‘streaming’ versions of baselines on iCubWorld 1.0 on (i) Class-iid, and (ii) Class-instance ordering. An empty plot in A-GEM indicates, we were unable to conduct experiment due to compatibility issues.

2. Problem Formulation

Online streaming learning (OSL) or streaming learning is the extreme case of online learning, where the data arrives sequentially one at a time, and the model needs to learn in a ‘single-pass’. Let us consider that we have a dataset $D$ with $T$ task sequences, i.e. $D = \{D_t\}_{t=1}^T$, where $D_t = \{(x_j, y_j)\}_{j=1}^{N_t}$. In streaming learning, it is assumed that $N_t = 1$ for all $t$, unlike incremental batch learning (IBL) and online learning approaches, where it is assumed $N_t \gg 1$ for all $t$. In addition, the model cannot loop over any part of the dataset, i.e., single-pass learning, and it can be evaluated immediately rather waiting for termination of training. This is in contrast to the incremental batch learning [1, 23] and online learning [33] approaches which allow visiting data-batches $D_t$ sequentially for multiple times and fine-tuning the network parameters with the buffer samples before inference, respectively. Furthermore, the data coming in streaming setting can be temporally contiguous, i.e., there could be a class based correlation, and the memory usage must be minimal. Finally, the model is evaluated in class incremental setting, such that, at test time, the label space is considered over all the classes observed so far.

3. Online Streaming Learning Framework

In this section, we introduce a ‘class incremental online-streaming learning’ framework (CIOSL) which trains a convolutional neural network (CNN) in streaming setting, as depicted in Figure 2. Formally, a limited number of samples are used to train the model offline during base initialization, $S^{(0)}$. Then in each incremental step, $\forall t > 0, S^{(t)}$, it observes a new example $(x_t, y_t)$, and the parameters are updated with a single step posterior computation. To avoid catastrophic forgetting, we use dual regularization by proposing implicit
and explicit regularization over the model parameters. Section 3.1 discusses the detailed regularization model. We also propose a novel online samples selection strategy for replay and buffer replacement which significantly improves model performance. We use a tiny episodic memory, and select only few informative past samples for replay instead of replaying the whole buffer, Section 3.2 and 3.3 are dedicated to the detailed discussion about the buffer policy.

Formally, we separate the CNN into two neural networks: (i) non-plastic feature extractor $G(\cdot)$ consisting the first few layers of the CNN, and (ii) plastic neural network $F(\cdot)$ consisting the final layers of the CNN. For a given input image $x$, the predicted class label is computed as: $y = F(G(x))$. We initialize the parameters of the feature extractor $G(\cdot)$ and keep it frozen throughout online streaming setting. We use a Bayesian-neural-network (BNN) [30] as the plastic network $F(\cdot)$, and optimize its parameters with sequentially coming data in streaming setting.

3.1. Learning in Online Streaming Scenario

Online updating naturally emerges from the Bayes’ rule: given the posterior $p(\theta|D_{1:t−1})$, whenever a new data $D_t = \{d_t\} = \{(x_t, y_t)\}$ comes in, we can compute a new posterior $p(\theta|D_{1:t})$ by combining the previous posterior and the new data-likelihood, i.e.,

$$p(\theta|D_{1:t}) \propto p(D_t|\theta)p(\theta|D_{1:t−1}),$$

where the old posterior is treated as the prior. However, for any complex model, exact Bayesian inference is not tractable, and an approximation is needed. A Bayesian neural network commonly approximates the posterior with a variational posterior $q(\theta)$ by maximizing the evidence-lower-bound (ELBO) [3]. However, optimizing the ELBO to approximate the posterior $p(\theta|D_{1:t})$ only with data $D_t$ can fail in streaming setting [13]. Furthermore, we evaluate the model’s performance in class incremental setting, such that, at test time the label space is considered over all the classes that have been observed till the time instance $t$. Moreover, training only with $D_t$ makes the model biased towards the new data or task and parameter regularization is not sufficient to overcome forgetting. To overcome these limitations, we include a ‘fixed-sized’ tiny episodic memory for storing a small representative subset ($\leq 5\%$) of all the observed samples. Instead of storing the raw input $x$, we store the embedding $z = G(x)$, where $z \in \mathbb{R}^d$. Storing the embeddings save a significant amount of memory, and also saves the model from doing a forward pass through the convolutional layers, making the training procedure computationally highly efficient.

During training, we select a subset of samples from memory instead of replaying the whole buffer, and replay them with the new data $D_t = \{d_t\} = \{(x_t, y_t)\}$. Therefore, the new posterior computation can be written as follows:

$$p(\theta|D_{1:t}) \propto p(D_t|\theta)p(D_{M,t}|\theta)p(\theta|D_{1:t−1}),$$

which we approximate with a variational posterior $q_t(\theta)$ as follows:

$$
\log p(y_t|\theta, G(x_t)) + \sum_{n=1}^{N'_t} \log p(y^{(n)}_{M,t}|\theta, z^{(n)}_{M,t})
- \lambda_1 \cdot KL (q_t(\theta) \| q_{t−1}(\theta))
\tag{3}
$$

where $D_{M,t} \subset M$ represents the subset of samples selected from the memory $M$ for replaying at time $t$, and $D_t$ represents the new data arriving at time $t$. Note that Eq. (1) is significantly different from VCL [31], where they assume multi-task/task-incremental learning setting with separate head networks, and incorporates the coreset samples only during explicit finetuning before inference.

The above minimization is equivalent to maximization of the evidence-lower-bound (ELBO):

$$
\mathcal{L}_t^1(\theta) = \mathbb{E}_{\theta \sim q_t(\theta)} [\log p(y_t|\theta, G(x_t))]
+ \sum_{n=1}^{N'_t} \mathbb{E}_{\theta \sim q_t(\theta)} [\log p(y^{(n)}_{M,t}|\theta, z^{(n)}_{M,t})]
- \lambda_1 \cdot KL (q_t(\theta) \| q_{t−1}(\theta))
\tag{2}
$$

where $D_{i,t} = \{(d_{i,t})\} = \{(x_{i,t}, y_{i,t})\}$, $D_{M,t} = \{d^{(n)}_{M,t}\}_{n=1}^{N'_t} = \{(z^{(n)}_{M,t}; y^{(n)}_{M,t})\}_{n=1}^{N'_t}$, $|D_{M,t}| = N'_t \ll |M|$, $D_{M,t} \subset M$, and $\lambda_1$ is a hyper-parameter.

While the KL-divergence minimization in Eq. (2) between prior and the posterior ensures minimal changes in the network parameters, initialization of the prior with the posterior at each time can introduce information loss in the network for a longer sequence of streaming data. We overcome such limitation with a functional/implicit regularizer, which encourages the network to mimic the output responses as produced in the past for the observed samples. Specifically, we minimize the KL-divergence between the class-probability scores obtained in past and current time $t$:

$$
\sum_{n=1}^{t−1} \mathbb{E}_{\theta \sim q_t(\theta)} [KL (softmax(h_n) \| F_{\theta}(G(x_n)))]
\tag{3}
$$

where $x_n$ and $h_n$ represents input examples and the logits obtained while training on $D_n$ respectively.

However, the objective in Eq. (3) requires the availability of the embeddings and the corresponding logits for all the past observed data. Since storing all the past data is not feasible, we only store the logits for all samples in memory.
uncertainty scores (negative samples) and \(N' \ll |M|\).

Under the mild assumptions of knowledge distillation [16], the optimization in Eq. (4) is equivalent to minimization of the Euclidean distance between the logits, and the optimization objective can be written as:

\[
L^2_t(\theta) = \sum_{n=1}^{N'_l} \mathbb{E}_{\theta \sim q_t(\theta)} [||h_n - f_\theta(z_n)||^2_2]
\]

where (i) \(f(\cdot)\) represents the network \(F(\cdot)\) without the softmax activation, and (ii) \(\lambda_2\) is a hyper-parameter.

Training the plastic network (BNN) \(F(\cdot)\) requires specification of \(q(\theta)\) and, in this work, we model \(\theta\) by stacking up the parameters (weights & biases) of the network \(F(\cdot)\). We use a Gaussian mean-field posterior \(q_t(\theta)\) for the network parameters, and choose the prior distribution, i.e., \(q_0(\theta) = p(\theta)\), as multivariate Gaussian distribution. We train the network \(F(\cdot)\) by maximizing the ELBO in Eq. (2) and minimizing the Euclidean distance in Eq. (5). For memory replay in Eq. (2), we select samples using strategies mentioned in Section 3.2, and we use uniform sample selection strategy to select samples from memory to be used in Eq. (5).

3.2. Informative Past Sample Selection For Replay

Uniform Sampling (Uni). In this approach, samples are selected uniformly from the memory.

Uncertainty-Aware Positive-Negative Sampling (UAPN). UAPN selects \(N'_1/2\) samples with the highest uncertainty scores (negative samples) and \(N'_1/2\) samples with the lowest uncertainty scores (positive samples). Empirically, we observe that this sample selection strategy results in the best performance. We measure the predictive uncertainty [5] for an input \(z\) with BNN \(F(\cdot)\) as follows:

\[
\Phi(z) \approx -\sum_c \left( \sum_k p(\hat{y} = c | z, \theta^k) \log \left( \sum_k p(\hat{y} = c | z, \theta^k) \right) \right)
\]

where \(p(\hat{y} = c | z, \theta^k)\) is the predicted softmax output for class \(c\) using the \(k\)-th sample of weights \(\theta^k\) from \(q(\theta)\). We use \(k = 5\) samples for uncertainty estimation.

Loss-Aware Positive-Negative Sampling (LAPN). LAPN selects \(N'_1/2\) samples with the highest loss-values (negative-samples), and \(N'_1/2\) samples with the lowest loss-values (positive-samples). Empirically we observe that the combination of most and least certain samples shows a significant performance boost since one ensures quality while the other ensures diversity for the memory replay.

3.3. Memory Buffer Replacement Policy

For memory replay, we use a ‘fixed-sized’ tiny episodic memory. However, in a lifelong learning setting, data may come indefinitely, implying that the capacity of the replay buffer will be quickly exhausted. To combat such an issue, we employ a replay buffer replacement policy which replaces a previously stored sample with a new instance if the buffer is full. Otherwise, the new instance is stored.

Loss-Aware Weighted Class Balancing Replacement (LAWCBR). In this approach, whenever a new sample comes in and the buffer is full, we remove a sample from the class with maximum number of samples present in the buffer \((y_r = \arg \max \text{ClassCount}(M))\). However, instead of removing an example uniformly, we weigh each sample of the majority class inversely w.r.t their loss \((w_i^y \propto \frac{1}{p_i})\) and use these weights as the replacement probability; the lesser the loss, the more likely to be removed.

Loss-Aware Weighted Random Replacement With A Reservoir (LAWRRR). In this approach, we propose a novel variant of reservoir sampling [41] to replace an existing sample with the new sample when the buffer is full. We weigh each stored sample inversely w.r.t the loss \((w_i \propto \frac{1}{p_i})\), and proportionally to the total number of examples of that class in which the sample belongs present in the buffer \((w_i \propto \text{ClassCount}(M, y_i))\). Whenever a new example satisfies the replacement condition of reservoir sampling, we combine these two scores and use that as the replacement probability; the higher the weight, the more likely to be replaced.

3.4. Making Sampling Strategies Online

Loss-aware sampling strategies require computing the loss-values of each stored sample in each incremental step, resulting in a computationally expensive learning process. To overcome this issue, we propose the following online update policy of the loss-values: (i) for each sample stored in memory, we keep the corresponding loss-value; (ii) whenever a sample is selected for memory reply, we replace the previously computed loss-value with the new loss-value at time \(t\); furthermore, we update the past logits with the newly computed logits, as changes in the loss-value reflect changes in the logits, too. To accommodate the online updating of loss-values, we propose to keep the loss-value in memory for each stored sample; however, since it is just a scalar value, the storage cost is negligible. To make uncertainty-aware sampling online, we store the uncertainty scores in memory and update them in a similar manner; the storage cost remains negligible, as it is another scalar value.

3.5. Feature Extractor

In this work, we separate the representation learning, i.e., learning the feature extractor \(G(\cdot)\), and the classifier learn-
ing, i.e., learning the plastic network \(F(\cdot)\). Similar to several existing continual learning approaches [14, 15, 20, 43], we initialize the feature extractor \(G(\cdot)\) with the weights learned through supervised visual representation learning [25], and keep them fixed throughout streaming learning. The motivation to use a pre-trained feature extractor is that the features learned by the first few layers of the neural networks are highly transferable and not specific to any particular task or dataset and can be applied to several different task(s) or dataset(s) [45]. Moreover, it is hard to learn generalized visual features, which can be used across all the classes [46] with having access to only a single example at each time instance.

In our experiments, for all the baselines along with CIOSL, we use Mobilenet-V2 [38] pre-trained on ImageNet [36] as the base architecture for the visual feature extractor. It consists of a convolutional base and a classifier network. We remove the classifier network and use the convolutional base as the feature extractor \(G(\cdot)\) to obtain embedding that is fed to the plastic network BNN \(F(\cdot)\). For details on the plastic network used for other baselines, refer to Section 5.3.

4. Related Work

Parameter-isolation-based approaches train different subsets of model parameters on sequential tasks. PNN [37], DEN [44] expand the network to accommodate the new task. PathNet [11], PackNet [29], Piggyback [28], and HAT [39] train different subsets of network parameters on each task.

Regularization-based approaches use an extra regularization term in the loss function to enable continual learning. LWF [27] uses knowledge distillation [16] loss to prevent catastrophic forgetting. EWC [23], IMM [26], and MAS [1] regularize by penalizing changes to the important weights of the network.

Rehearsal-based approaches replay a subset of past training data during sequential learning. iCaRL [34], SER [17], and TinyER [8] use memory replay when training on a new task. DER [4] uses knowledge distillation and memory replay while learning a new task. DGR [40], MeRGAN [42], and CloGAN [35] retain the past task(s) distribution with a generative model and replay the synthetic samples during incremental learning. Our approach also leverages memory replay from a tiny episodic memory; however, we store the feature-maps instead of raw inputs.

Variational Continual Learning (VCL) [31] leverages Bayesian inference to mitigate catastrophic forgetting. However, the approach, when naively adapted, performs poorly in the streaming learning setting. Additionally, it also needs task-id during inference. Furthermore, the explicit finetuning with the buffer samples (coreset) before inference violates the single-pass learning constraint. Moreover, it still does not outperform our approach even with the finetuning. More details are given in the appendix.

REMINd [15] is a recently proposed rehearsal-based lifelong learning approach, which combats catastrophic forgetting in online streaming setting. While it follows a setting close to the one proposed, the model stores a large number of past examples compared to the other baselines; for example, iCaRL [34] stores 10K past examples for the ImageNet experiment, whereas REMIND stores 1M past examples. Further, it actually uses a lossy compression to store past samples, which is merely an engineering technique, not an algorithmic improvement, and can be used by any continual learning approach. For more details, please refer to the appendix.

5. Experiments

5.1. Baselines And Compared Methods

The proposed approach follows the ‘online streaming setting’; to the best of our knowledge, recent works ExStream [14] and REMIND [15] are the only methods that follow this setting. We compare our approach against these strong baselines. We also compare our model with (i) a network trained with a sample one at a time (Fine-tuning/lower-bound) and (ii) a network trained offline, assuming all the data is available (Offline/upper-bound). Also, for rigorous comparison, we choose recent popular ‘batch’ and ‘online’ continual learning methods, such as EWC [23], MAS [1], VCL with/without coreset [31], Coreset Only [10], TinyER [8], GDumb [33] and A-Gem [7]. For a fair comparison, we train all the methods in an online streaming setting, i.e., one sample at a time. ‘VCL w/ C’ and ‘Coreset only’ are both trained in a streaming manner; however, the network is fine-tuned with the stored samples before inference. Also, GDumb stores samples in memory and fine-tunes the network with them before inference, while fine-tuning is prohibited in ‘streaming setting’. Therefore, VCL and GDumb have an extra advantage compared to the true ‘streaming learning’ approaches. Still, CIOSL outperforms these approaches by a significant margin. More details are given in the appendix.

5.2. Datasets, Data Orderings And Metrics

Datasets. To evaluate the efficacy of the proposed model we perform extensive experiments on four standard datasets: CIFAR10, CIFAR100 [24], ImageNet100, and iCubWorld-1.0 [9]. CIFAR10 and CIFAR100 are standard classification datasets with 10 and 100 classes, respectively. ImageNet100 is a subset of ImageNet-1000 (ILSVRC-2012) [36] containing randomly chosen 100 classes, with each class containing 700-1300 training samples and 50 validation samples, which are used for testing. iCubWorld 1.0 is an object recognition dataset which contains the sequences of video frames, with each containing a single object. There are 10 classes, with
Table 1. Categorization of the baseline approaches depending on the underlying simplifying assumptions they impose. V.C.SL: Violates Constraints of Streaming Learning

| Method     | Learning Type | Fine-tunes | V.C.SL | Regularize | Memory |
|------------|---------------|------------|--------|------------|--------|
| EWC        | Batch         | ×          | ×      | ✓          | ×      |
| MAS        | Batch         | ×          | ×      | ✓          | ×      |
| VCL        | Batch         | ×          | ×      | ✓          | ×      |
| VCL w/C    | Batch         | ×          | ×      | ✓          | ×      |
| Coreset    | Online        | ✓          | ✓      | ×          | ×      |
| GDumb      | Online        | ✓          | ×      | ✓          | ×      |
| TinyER     | Online        | ×          | ×      | ✓          | ×      |
| A-GEM      | Online        | ×          | ×      | ✓          | ×      |
| ExStream   | Streaming     | ×          | ×      | ✓          | ×      |
| REMIND     | Streaming     | ×          | ×      | ✓          | ×      |
| Ours       | Streaming     | ×          | ×      | ✓          | ×      |

Each containing 3 different object instances with 200-201 images each. Overall, each class contains 600-602 samples for training and 200-201 samples for testing. Technically, iCubWorld 1.0 is ideal dataset for streaming learning, as it requires learning from temporally ordered image sequences, i.e., non-i.i.d images.

Evaluation Over Different Data Orderings. The proposed approach is robust to the various streaming learning setting; we evaluate the model’s streaming learning ability with the following four [14, 15] challenging data ordering schemes: (i) ‘streaming iid’: where the data-stream is organized by the randomly shuffled samples from the dataset, (ii) ‘streaming class iid’: where the data-stream is organized by the samples from one or more classes, these samples are shuffled randomly, (iii) ‘streaming instance’: where the data-stream is organized by temporally ordered samples from different object instances, and (iv) ‘streaming class instance’: where the data-stream is organized by the samples from different classes, the samples within a class are temporally ordered based on different object instances. Only iCubWorld dataset contains the temporal ordering therefore ‘streaming instance’, and ‘streaming class instance’ setting evaluated only on the iCubWorld dataset. Please refer to the appendix for more details.

Metrics. For evaluating the performance of the streaming learner, we use Ωall metric, similar to [14, 15, 21], where Ωall represents normalized incremental learning performance with respect to an offline learner: Ωall = 1/T Σt=1 T αall t , where T is the total number of testing events, αt is the performance of the incremental learner at time t, and αoffline,t is the performance of a traditional offline model at time t.

5.3. Implementation Details

In all the experiments, models are trained with one sample at a time. For a fair comparison, a similar network structure is used throughout all the models. For all methods, we use fully connected single-head networks with two hidden layers as the plastic network F(·), where each layer contains 256 nodes with ReLU activations; for VCL w/ or w/o Coreset, Coreset only and CIOSL, F(·) is a BNN, whereas for all other methods F(·) is a deterministic network. For a fair comparison, we store the same number of past examples for all replay-based approaches. For REMIND, we compress and store the feature-maps with Faiss [19] product quantization (PQ) implementation with s = 32 sub-vectors and codebook size c = 256.

We store the feature-map in memory for all the other methods, including our approach CIOSL. In addition, CIOSL also store the corresponding logits, loss-values, and uncertainty-scores. The capacity of our replay buffer is mentioned in Table 3. For memory-replay, we use ‘uncertainty-aware positive-negative’ sampling strategy (discussed in Sec:3.2) throughout all data-orderings, except for ‘streaming-i.i.d’ ordering, we use ‘uniform’ sampling. We use ‘loss-aware weighted random replacement with a reservoir’ sampling strategy as memory replacement policy for all the experiments. We store the same number of past examples in memory across all methods. For memory-replay, we use N1 = 16 past samples throughout all experiments across CIOSL, AGEM, TinyER, ExStream and REMIND. For knowledge-distillation, CIOSL use N2 = 16 samples at any time step t. We set the hyper-parameter λ1 = 1 and λ2 = 0.3 across all experiments. For EWC, we set hyper-parameter λ = 500, and for MAS, we set hyper-parameter λ = 1. We repeated each experiments for 10 times with different permutations of the data, and reported the results by taking average of 10 runs. Please refer to the appendix for more details.

5.4. Results

The detailed results of CIOSL over various experimental settings along with the strong baseline methods are shown in Table 2. We can clearly observe that CIOSL consistently outperforms the baseline by a significant margin. The proposed model is also robust to the different streaming learning scenarios compared to the baselines. We repeat our experiment ten times, because of the space constraint, we omit standard-deviations and included in the appendix. We do not consider GDumb as the best-performing method, even when it achieves higher accuracy for the class-iid since it finetunes the network with the stored samples and makes the learning algorithm a ‘two-step’ process, which is prohibited in streaming-learning. We observe that ‘batch-learning’ methods severely suffer from catastrophic forgetting. Moreover, replay-based ‘online-learning’ method such as AGEM also suffer from information loss badly. Furthermore, GDumb, even with finetuning before each inference, cannot achieve the best accuracy in all the experiments. CIFAR10/100, ImageNet are the standard classification datasets, while iCubWorld 1.0 is a challenging dataset that evaluates the models in more realistic scenarios or data-orderings. Particularly,
Table 2. $\Omega_{\text{all}}$ results. For each experiment, the method with best performance in ‘streaming-setting’ is highlighted in Bold. The reported results are average over 10 runs with different permutations of the data. Offline model is trained only once. Offline $= \frac{1}{T} \sum_{t=1}^{T} \alpha_{\text{offline}, t}$. where $T$ is the total number of testing events. ‘-’ indicates experiments we were unable to run, because of compatibility issues. Methods in Red use fine-tuning before inference, which violates ‘single-pass’ learning constraint.

| Dataset       | Buffer Capacity | CIFAR10 | CIFAR100 | ImageNet100 | iCubWorld 1.0 |
|---------------|-----------------|---------|----------|-------------|--------------|
| Training-Set  | Size            |         |          |             |              |
|               |                 | 1000    | 1000     | 1000        | 180          |
|               | 50000           | 50000   | 127778   | 6002        |

Table 3. Memory buffer capacity used for various datasets.

class-instance ordering and instance ordering require the learner to learn from temporally ordered video frames one at a time. From Table 2, we observe that CIOSL obtain up to 8.58% and 6.26% improvement over the state-of-the-art streaming learning approaches. In fact, in most of the scenarios, CIOSL is very close to the upper bound performance, i.e., when the model is trained in a fully offline fashion.

For completeness, we train ‘Our-method’ in ‘batch’ and ‘online’ learning setting to determine its effectiveness and compatibility in these settings. In Figure 1, we compare $\Omega_{\text{all}}$ results of various baseline with our-method CIOSL. We observe that CIOSL outperforms the baselines by a significant margin on both class-i.i.d and class-instance ordering on iCubWorld 1.0 dataset. This implies that a method trained in the extreme setting can be thought of as a universal method for all lifelong learning settings with the widest possible applicability. More details are given in the appendix.

6. Ablation Study

We perform extensive ablation to show the importance of the different components. The various ablation experiments validate the significance of the proposed components.

Significance Of Different Sampling Strategies. In Table 4, we compare the performance of CIOSL while using various sampling strategies and memory replacement policies. We observe that for the buffer replacement, LAWR

Table 4. $\Omega_{\text{all}}$ Results. For each experiment, the method with best performance is highlighted in Bold.

Choice Of Hyperparameter ($\lambda_2$). Figure 4 shows the effect of changing the knowledge-distillation loss weight $\lambda_2$ on the final $\Omega_{\text{all}}$ accuracy for iCubWorld 1.0 on instance and class-instance ordering, while using different sampling strategies and buffer replacement policies. We observe the best model performance for $\lambda_2 = 0.3$, and we use this value of $\lambda_2$ for all our experiments. We provide detailed ablation on $\lambda_2$ in the appendix.

Significance Of Knowledge-Distillation Loss. Figure 4 with $\lambda_2 = 0.0$ represents the model without knowledge distillation. We can observe that the model performance significantly degrades without knowledge distillation. Therefore, knowledge distillation is a key component to the model’s performance. More details are given in the appendix.

Choice Of Buffer Capacity. We perform an ablation for the different buffer capacities, i.e., $|M|$. The results are shown in Figure 5. It is evident that, with the longer se-
Results

(ii) class-instance ordering on iCubWorld 1.0.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In Proceedings of the European Conference on Computer Vision (ECCV), pages 139–154, 2018. 1, 2, 5, 10, 12

[2] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. Selfless sequential learning. arXiv preprint arXiv:1806.05421, 2018. 10

[3] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. arXiv preprint arXiv:1505.05424, 2015. 3, 10

[4] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. arXiv preprint arXiv:2004.07211, 2020. 5

[5] Lucy R Chai. Uncertainty estimation in bayesian neural networks and links to interpretability. Master of Philosophy (University of Cambridge), 2018. 4

[6] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In Proceedings of the European Conference on Computer Vision (ECCV), pages 532–547, 2018. 10

[7] Arslan Chaudhry, Marc’ Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. arXiv preprint arXiv:1812.00420, 2018. 1, 5, 12

[8] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019. 5, 12

[9] Sean Fanello, Carlo Ciliberto, Matteo Santoro, Lorenzo Natale, Giorgio Metta, Lorenzo Rosasco, and Francesca Odone. icub world: Friendly robots help building good vision datasets. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 700–705, 2013. 5, 13, 15, 17

[10] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. arXiv preprint arXiv:1805.09733, 2018. 5, 10, 12

[11] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. arXiv preprint arXiv:1701.08734, 2017. 5

[12] Robert M French. Catastrophic forgetting in connectionist networks. Trends in cognitive sciences, 3(4):128–135, 1999. 1, 10, 13, 15

[13] Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. In International Conference on Machine Learning, pages 1744–1753. PMLR, 2018. 3

[14] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In 2019 International Conference on Robotics and Automation (ICRA), pages 9769–9776. IEEE, 2019. 1, 2, 5, 6, 10, 11, 12, 13

[15] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. arXiv preprint arXiv:1910.02509, 2019. 1, 5, 6, 10, 11, 13

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015. 4, 5

7. Conclusion

Streaming continual learning (SCL) is the most challenging and realistic framework for continual learning; most of the recent promising models for the CL are unable to handle this above setting. Our work proposes a dual regularization and loss-aware sample replay to handle the SCL scenario. The proposed model is highly efficient since it learns a joint likelihood from the current and replay samples without leveraging any external finetuning. Also, to improve the training efficiency further, the proposed model selects a few most informative samples from the buffer instead of using the entire buffer for the replay. We have conducted a rigorous experiment over several challenging datasets and showed that CIOSL outperforms state-of-the-art approaches in this setting by a significant margin. To disentangle the importance of the various components, we perform extensive ablation studies and observe that the proposed components are essential to handle the SCL setting.
A. Preliminaries

A.1. ‘Class Incremental Learning’ V/S ‘Task Incremental/Multi-Task Learning’

‘Class incremental learning’ [6, 14, 15, 34], is a challenging variant of lifelong learning, where the classifier needs to learn to discriminate between different class labels from different tasks. The key distinction between ‘class incremental learning’ and ‘task incremental/multi-task learning’ [1, 2, 23, 31], lies in how the classifier’s accuracy is evaluated at the test time. In ‘class incremental learning’, at the test time, the task identifier $t$ is not specified, and the accuracy is computed over all the observed classes with $\frac{1}{C}$ chance, where $C$ is the total number of classes accumulated so far. However, in ‘task incremental learning’, the task identifier $t$ is known.

For example, consider MNIST divided into 5 tasks: $\{\{0, 1\}, \ldots, \{8, 9\}\}$, which are used for sequential learning of a classifier. Then, at the end of 5-th task, in ‘task incremental setting’, the classifier needs to predict a class out of $\{8, 9\}$ only. However, in ‘class incremental setting’, a class label is predicted over all the ten classes that is observed so far, i.e., $\{0, \ldots, 9\}$ with $\frac{1}{10}$ chance for each class.

A.2. Variational Continual Learning (VCL)

Variational Continual Learning (VCL) [31] is a recently proposed continual learning approach that mitigates catastrophic forgetting in neural networks in a Bayesian framework. It sets the posterior of parameters distribution as the prior before training on the next task, i.e., $p_t(\theta) = q_{t-1}(\theta)$, the new task reuses the previous task’s posterior as the new prior. VCL solves the following KL divergence minimization problem while training on task $t$ with the new data $D_t$:

$$q_t(\theta) = \underset{q \in Q}{\operatorname{arg\ min}} \quad \text{KL} \left( q(\theta) \mid\mid \frac{1}{Z_t} q_{t-1}(\theta) \cdot p(D_t | \theta) \right) \quad (7)$$

While offering a principled way of continual learning, VCL follows task incremental / multi task learning setting, and uses ‘task specific head networks’, for each task $t$, such that, $p(\theta | D_{1:t}) = p(\theta_1 | D_1) \cdot p(\theta_S | D_{1:t})$, where $\theta = \{\theta_S, \theta_t\}$, $\theta_1$ is shared between all the tasks, whereas $\theta_t$ kept fixed after training on task $t$. This configuration prohibits knowledge transfer across tasks, and results in a poor accuracy in class incremental setting [10] for both VCL with or without Coreset.

VCL with Coreset [31] withheld some data points from the task data before training and keeps them in a coreset. These data points are not used for the network training and are only used for finetuning the network before each inference. However, in online streaming learning finetuning the network at any time is prohibited, as it makes the training process a two-step learning process instead of single-pass learning. Furthermore, the coreset is created by sampling data points from the entire task data, whereas in online streaming setting, each instance arrives one at a time. Finally, the performance of VCL with Coreset is heavily dependent on the finetuning with the withheld samples, i.e., coreset samples, before inference [10], and still not comparable enough to our proposed method (CIOSL).

A.3. REMIND

REMIND [15] is a recently proposed replay-based lifelong learning approach which combats catastrophic forgetting [12] in deep neural network in online-streaming setting. While following such a challenging setting, it separates the convolutional neural network into two networks: (i) a frozen feature extractor and (ii) a plastic neural network. Learning involves the following steps: (i) compression of each new input using product quantization (PQ) [18], (ii) reconstruction of the previously stored compressed representations using PQ, and (iii) mixing the reconstructed past examples with the new input and updating the parameters of the plastic layers of the network.

While it offers a principled way to combat catastrophic forgetting and achieves state-of-the-art performance, there are few concerns that can be limiting in the continual learning setup. It stores considerably a large number of past examples compared to the baselines; for example, iCaRL [34] stores 10K past examples for ImageNet experiment whereas REMIND stores 1M past examples. Furthermore, REMIND actually uses a lossy compression method (PQ) to store the past samples, which is merely an engineering technique far from any algorithmic improvement and can be used by any lifelong learning approach.

A.4. Bayesian Neural Network

Bayesian neural networks [30] are discriminative models, which extend the standard deep neural networks with Bayesian inference. The network parameters are assumed to have a prior distribution, $p(\theta)$, and it infers the posterior given the observed data $D$, that is, $p(\theta | D)$. However, the exact posterior inference is computationally intractable for any complex models, and an approximation is needed. One such scheme is ‘Bayes-by-Backprop’ [3]. It uses a mean-field variational posterior $q(\theta)$ over the network parameters and uses reparameterization-trick [22] to sample from the posterior, which are then used to approximate the evidence lower bound (ELBO) via Monte-Carlo sampling.

In our proposed method (CIOSL), we have used a Bayesian neural network (BNN) as the plastic network $F(\cdot)$. We have discussed training the plastic network (BNN) $F(\cdot)$ with a single step posterior update without catastrophic forgetting [12] in Section 3.1 in the main paper.
Table 5. \( \Omega_{\text{all}} \) with their associated standard deviations for ‘batch’, ‘online’, and ‘streaming’ versions of baselines on iCubWorld 1.0 on (i) class-i.i.d, and (ii) class-instance ordering. ‘-’ indicates experiments we were unable to run, because of compatibility issues.

| Method       | CIFAR10         | CIFAR100        | ImageNet100     | CIFAR10         | CIFAR100        | ImageNet100     |
|--------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|              | class-i.i.d     | class-instance  | class-i.i.d     | class-instance  | class-i.i.d     | class-instance  |
| VCL          | 0.1175 ± 0.0000 | 0.0180 ± 0.0035 | 0.0127 ± 0.0029 | 0.3477 ± 0.0003 | 0.1277 ± 0.0022 | 0.1223 ± 0.0052 |
| VCL w/ C/    | 0.1175 ± 0.0000 | 0.0180 ± 0.0035 | 0.0127 ± 0.0029 | 0.3477 ± 0.0003 | 0.1277 ± 0.0022 | 0.1223 ± 0.0052 |
| A-GEM        | 0.3446 ± 0.0003 | 0.1292 ± 0.0037 | 0.1225 ± 0.0039 | 0.3446 ± 0.0003 | 0.1292 ± 0.0037 | 0.1225 ± 0.0039 |
| TinyER       | 0.3446 ± 0.0003 | 0.1292 ± 0.0037 | 0.1225 ± 0.0039 | 0.3446 ± 0.0003 | 0.1292 ± 0.0037 | 0.1225 ± 0.0039 |
| REMIND       | 0.3716 ± 0.0501 | 0.1414 ± 0.0224 | 0.1259 ± 0.0122 | 0.3716 ± 0.0501 | 0.1414 ± 0.0224 | 0.1259 ± 0.0122 |
| Ours         | 0.3684 ± 0.0442 | 0.1432 ± 0.0256 | 0.1273 ± 0.0182 | 0.3684 ± 0.0442 | 0.1432 ± 0.0256 | 0.1273 ± 0.0182 |

Table 6. \( \Omega_{\text{all}} \) results with their associated standard deviations. For each experiment, the method with best performance in ‘streaming-setting’ is highlighted in **Bold**. The reported results are average over 10 runs with different permutations of the data. Offline model is trained only once. Offline = \( \frac{1}{T} \sum_{t=1}^{T} \alpha_{\text{offline},t} \), where T is the total number of testing events. ‘-’ indicates experiments we were unable to run, because of compatibility issues. Methods in Red use fine-tuning before inference, which violates ‘single-pass’ learning constraint.
Table 7. \( \Omega_{\text{all}} \) results with their associated standard deviations. For each experiment, the method with best performance in ‘streaming-setting’ is highlighted in **Bold**. The reported results are average over 10 runs with different permutations of the data. Offline model is trained only once. Offline = \( \frac{1}{T} \sum_{t=1}^{T} \alpha_{\text{offline}} \), where \( T \) is the total number of testing events. '-' indicates experiments we were unable to run, because of compatibility issues. Methods in Red use fine-tuning before inference, which violates ‘single-pass’ learning constraint.

| Method                  | iid      | Class-iid | Instance | Class-instance |
|-------------------------|----------|-----------|----------|----------------|
| Fine-Tune               | 0.1369 ± 0.0184 | 0.3893 ± 0.0534 | 0.1307 ± 0.0000 | 0.3485 ± 0.0022 |
| EWC                     | -        | 0.3790 ± 0.0419 | -        | 0.3487 ± 0.0034 |
| MAS                     | -        | 0.3912 ± 0.0613 | -        | 0.3486 ± 0.0019 |
| VCL                     | -        | 0.3806 ± 0.0527 | -        | 0.3473 ± 0.0025 |
| VCL w/ C/               | -        | 0.3948 ± 0.0558 | -        | 0.4705 ± 0.0165 |
| Coreset                 | -        | 0.3994 ± 0.0922 | -        | 0.4669 ± 0.0251 |
| GDumb                   | 0.8993 ± 0.0413 | **0.9660 ± 0.0201** | 0.6715 ± 0.0540 | 0.7908 ± 0.0329 |
| A-GEM                   | 0.1311 ± 0.0000 | 0.4047 ± 0.0632 | 0.1309 ± 0.0003 | 0.3489 ± 0.0030 |
| TinyER                  | 0.9590 ± 0.0378 | 0.9069 ± 0.0297 | 0.8726 ± 0.0649 | 0.8215 ± 0.0341 |
| ExStream                | 0.9235 ± 0.0584 | 0.8820 ± 0.0285 | 0.8954 ± 0.0542 | 0.8727 ± 0.0229 |
| REMIND                  | 0.9260 ± 0.0311 | 0.8553 ± 0.0349 | 0.8157 ± 0.0600 | 0.7615 ± 0.0319 |
| **Ours**                | **0.9716 ± 0.0141** | **0.9480 ± 0.0215** | **0.9580 ± 0.0298** | **0.9585 ± 0.0223** |

4. **VCL with Coreset** [31]: This method is the same as pure VCL as mentioned above, except, at the end of training on each task, the network is finetuned with the coreset samples. We adapted the coreset selection in online streaming setting and stored data points in coreset in an online manner.

5. **Coreset Only** [10]: This method is exactly similar to VCL with Coreset [31], except the prior which is used for variational inference is the initial prior each time, i.e., it is not updated with the previous posterior before training on a new task.

6. **GDumb** [33]: It is an online learning method. It stores data points with a greedy sampler and retraining the network from scratch each time with stored samples before inference.

7. **A-GEM** [7]: It is another online learning approach. It uses past task data stored in memory to build an optimization constraint to be satisfied by each new update. If the gradient violates the constraint, then it is projected such that the constraint is satisfied.

8. **TinyER** [8]: It stores past task data points in a tiny episodic memory and replays them with the current training data to enable continual learning.

9. **ExStream** [14]: It is an online streaming learning method, which uses memory replay to enable continual learning. It maintains buffers of prototypes to store the input vectors. Once the buffer is full, it combines the
two nearest prototypes in the buffer and stores the new input vector.

10. **REMINd** [15]: Similar to ExStream, it is another *streaming learning* method, which enables lifelong learning with memory replay. For more details on REMIND, please refer to Section A.3.

11. **Fine-tuning**: It is a streaming learning baseline and serves as the lower bound on the network’s performance. In this scenario, the network parameters are fine-tuned with one instance through the whole dataset for a single epoch.

12. **Offline**: It serves as the upper bound on the network’s performance, where the network is trained in the traditional way; the complete dataset is divided into multiple batches, and the network loops over them multiple times.

**Note:** In *online streaming setting*, finetuning the network with the stored samples is prohibited, as it violates the *single-pass* learning constraint. ‘VCL with Coreset’, ‘Core-set Only’, and ‘GDumb’ finetune the network parameters before inference; therefore, these methods have an extra advantage compared to true *streaming learning* approaches, and they violate the *single-pass* learning constraint. Therefore these methods cannot be considered as the best-performing methods even when they achieve better final accuracy as these methods are not true *streaming learning* method.

Table 8 categorizes the baselines according to the underlying assumptions that they impose. For baselines which finetune the network before inference and violate the constraint of *streaming learning*, such as single pass learning constraint, have been marked in red in the corresponding column.

### D. Ablation Study Additional Results

In this section, we provide additional results for ablation studies, which we could not provide in the main paper due to space constraints.

- **ImageNet100.** In Table 9, we compare the final $\Omega_{all}$ accuracy of CIOSL for (i) i.i.d, and (ii) class-i.i.d ordering on ImageNet100 dataset while using different memory replacement policy and past sample selection strategies.
- **CIFAR10/100.** Table 10 compares the final $\Omega_{all}$ accuracy of the proposed model (CIOSL) for (i) i.i.d and (ii) class-i.i.d ordering on CIFAR10 and CIFAR100 respectively while using different values for the knowledge-distillation hyper-parameter $\lambda_2$, and different memory replacement policies and various sample selection strategies.

- **iCubWorld 1.0.** Table 11 and Table 12 compares the final $\Omega_{all}$ accuracy of CIOSL for (i) i.i.d, (ii) class-i.i.d, (iii) instance, and (iv) class-instance ordering on iCubWorld 1.0 dataset while using different knowledge-distillation hyper-parameter $\lambda_2$ and different sampling strategies. For memory replacement policy, Table 11 uses ‘loss-aware weighted class balancing replacement (LAWCBR)’ strategy, whereas Table 12 uses ‘loss-aware weighted random replacement with a reservoir (LAWRRR)’ strategy.

### E. ImageNet-100

In this paper, we used a subset of ImageNet-1000 (ILSVRC-2012) [36] that contains randomly chosen 100 classes. To ease a relevant study, we release the list of these 100 classes that we used to evaluate the streaming learner’s performance in our experiments, as mentioned in Table 14.

### F. Additional Implementation Details

In this section, we provide some additional implementation details, which we could not provide in the main paper due to space constraints.

We use Mobilenet-V2 [38] pre-trained on ImageNet [36] available in PyTorch [32] TorchVision package as the base architecture for the feature extractor $G(\cdot)$. We use the convolutional base of Mobilenet-V2 [38] as the feature extractor $G(\cdot)$ to obtain embeddings from the raw pixels; we keep it frozen throughout the streaming learning. We use *uniform sampling* and knowledge-distillation hyper-parameter $\lambda_2 = 0.2$ for *online learning* and *batch learning* experiments (in Table 5). We provide the parameter settings for the proposed method (CIOSL) and the offline models in Table 13.

### G. Evaluation Over Different Data Orderings Additional Details

The proposed approach (CIOSL) is robust to various *streaming learning scenarios* that can induce catastrophic forgetting [12]. We evaluate the model’s streaming learning ability with the four challenging data ordering [14, 15] schemes: (i) ‘streaming iid’, (ii) ‘streaming class iid’, (iii) ‘streaming instance’, and (iv) ‘streaming class instance’. We described this four data ordering schemes in more detail in Section 5.2 in the main paper.

**Note:** Only iCubWorld 1.0 [9] dataset contains the temporal ordering, therefore, ‘streaming instance’ ‘streaming class instance’ setting evaluated only on the iCubWorld 1.0 dataset.

In the below, we describe the following: (i) how the base initialization is performed, and (ii) how the network is trained in *streaming setting* according to various data ordering schemes on different datasets.
Method | Learning Type | Fine-tunes | Violates Constraints Of Streaming Learning | Regularize | Memory
---|---|---|---|---|---
EWC | Batch | x | | ✓ | x
MAS | Batch | x | | ✓ | x
VCL | Batch | x | | ✓ | x
VCL w/ C/ | Batch | ✓ | ✓ | ✓ | ✓
Coreset | Online | ✓ | ✓ | x | ✓
GDumb | Online | ✓ | ✓ | x | ✓
TinyER | Online | x | x | x | ✓
A-GEM | Online | x | x | ✓ | ✓
ExStream | Streaming | x | x | x | ✓
REMIND | Streaming | x | x | x | ✓
Ours | Streaming | ✓ | ✓ | ✓ | ✓

Table 8. Categorization of the baseline approaches depending on the underlying simplifying assumptions they impose.

| Memory Replacement | Sample Selection | Imagenet100 |
|---|---|---|
| | iid | Class-iid |
| LAWCBR | Uni | 0.9582 ± 0.0037 | 0.9014 ± 0.0073 |
| | UAPN | 0.9327 ± 0.0052 | 0.9115 ± 0.0081 |
| | LAPN | 0.9253 ± 0.0115 | 0.9122 ± 0.0091 |
| LAWRRR | Uni | 0.9640 ± 0.0060 | 0.8643 ± 0.0127 |
| | UAPN | 0.9575 ± 0.0035 | 0.9171 ± 0.0073 |
| | LAPN | 0.9575 ± 0.0047 | 0.9112 ± 0.0075 |

Table 9. \( \Omega_{al} \) Results as a function of different memory replacement policies and sample selection strategies for (i) i.i.d, and (ii) class-i.i.d ordering on ImageNet100.

G.1. CIFAR10

CIFAR10 [24] is a standard image classification dataset. It contains 10 classes with each consists of 5000 training images and 1000 testing images. Since, it does not contain any temporally ordered image sequence, we use CIFAR10 to evaluate the streaming learner’s ability in streaming i.i.d and streaming class-i.i.d orderings.

- streaming i.i.d: For the base initialization, we randomly select 2% samples from the dataset and train the model in offline manner. Then we randomly shuffle the remaining samples and train the model incrementally with these samples by feeding one at a time in a streaming manner.

- streaming class-i.i.d: In base initialization, the model is trained in a typical offline mode with the samples from the first two classes. Then, in each incremental step, we select the samples from the next two classes, which are not included earlier. These samples are randomly shuffled and fed into the model in a streaming manner.

G.2. CIFAR100

CIFAR100 [24] is another standard image classification dataset. It contains 100 classes with each consists of 500 training images and 100 testing images. We use CIFAR100 to evaluate the model’s ability in streaming i.i.d and streaming class-i.i.d orderings.

- streaming i.i.d: In this setting, we follow the similar approach as mentioned for the CIFAR10 dataset, with the only exception is that the base initialization is performed with 10% randomly chosen samples, and the remaining samples are used for streaming learning.

- streaming class-i.i.d: This approach also follows the similar approach as mentioned for the CIFAR10 dataset. However, in each incremental step, including the base initialization, we use samples from 10 classes. For the base initialization, we select samples from the first ten classes, and in each incremental step, we select samples from the succeeding ten classes which are not observed earlier.

G.3. ImageNet100

ImageNet100 is a subset of ImageNet-1000 (ILSVRC-2012) [36] that contains randomly chosen 100 classes, with each classes containing 700 – 1300 training samples and 50 validation samples. Since, for test samples, we do not have the ground truth labels, we use the validation data for testing the model’s accuracy. We provide more details on ImageNet100 in Section E.

We use ImageNet100 dataset to evaluate the model’s ability in streaming i.i.d and streaming class-i.i.d orderings.

- streaming i.i.d: In this case, we follow the similar approach as mentioned for CIFAR100 streaming i.i.d ordering.
| $\lambda_2$ | Memory Replacement | Sample Selection | iid | class-iid |
|---|---|---|---|---|
| 0.2 | LAWCBR | Uni | 0.9542 ± 0.0053 | 0.8135 ± 0.0054 | 0.8942 ± 0.0062 | 0.7343 ± 0.0131 |
|  |  | UAPN | 0.9084 ± 0.0121 | 0.4760 ± 0.0136 | 0.8957 ± 0.0125 | 0.6448 ± 0.0257 |
|  |  | LAPN | 0.8462 ± 0.0414 | 0.3834 ± 0.0335 | 0.8797 ± 0.0149 | 0.5332 ± 0.0310 |
|  | LAWRRR | Uni | 0.9584 ± 0.0035 | 0.8617 ± 0.0091 | 0.8792 ± 0.0104 | 0.7221 ± 0.0149 |
|  |  | UAPN | 0.9567 ± 0.0031 | 0.8366 ± 0.0107 | 0.8978 ± 0.0110 | 0.7589 ± 0.0185 |
|  |  | LAPN | 0.9530 ± 0.0037 | 0.8273 ± 0.0141 | 0.8986 ± 0.0121 | 0.7478 ± 0.0191 |
| 0.3 | LAWCBR | Uni | 0.9529 ± 0.0062 | 0.8134 ± 0.0077 | 0.8970 ± 0.0088 | 0.7369 ± 0.0106 |
|  |  | UAPN | 0.9145 ± 0.0071 | 0.5096 ± 0.0088 | 0.8944 ± 0.0093 | 0.6836 ± 0.0231 |
|  |  | LAPN | 0.9046 ± 0.0152 | 0.4376 ± 0.0220 | 0.8798 ± 0.0230 | 0.6275 ± 0.0291 |
|  | LAWRRR | Uni | 0.9579 ± 0.0040 | 0.8679 ± 0.0057 | 0.8838 ± 0.0088 | 0.7307 ± 0.0122 |
|  |  | UAPN | 0.9567 ± 0.0031 | 0.8542 ± 0.0066 | 0.8991 ± 0.0089 | 0.7724 ± 0.0188 |
|  |  | LAPN | 0.9538 ± 0.0044 | 0.8453 ± 0.0120 | 0.9024 ± 0.0116 | 0.7573 ± 0.0193 |

Table 10. $\Omega_{\alpha}$ Results as a function of knowledge-distillation hyper-parameter $\lambda_2$ and different memory replacement policies and sample selection strategies for (i) i.i.d ordering, and (ii) class-i.i.d ordering on CIFAR10 and CIFAR100 datasets.

| $\lambda_2$ | Sample Selection | iid | Class-iid | Instance | Class-instance |
|---|---|---|---|---|---|
| 0.0 | Uni | 0.9431 ± 0.0418 | 0.9105 ± 0.0333 | 0.8414 ± 0.0541 | 0.8259 ± 0.0316 |
|  | UAPN | 0.8775 ± 0.0753 | 0.8863 ± 0.0529 | 0.6777 ± 0.0764 | 0.7711 ± 0.0574 |
|  | LAPN | 0.8975 ± 0.0697 | 0.8675 ± 0.0498 | 0.7576 ± 0.0739 | 0.7524 ± 0.0655 |
| 0.1 | Uni | 0.9885 ± 0.0245 | 0.9163 ± 0.0237 | 0.9257 ± 0.0299 | 0.8369 ± 0.0329 |
|  | UAPN | 0.9781 ± 0.0318 | 0.9167 ± 0.0263 | 0.9124 ± 0.0252 | 0.8627 ± 0.0285 |
|  | LAPN | 0.9779 ± 0.0206 | 0.9224 ± 0.0332 | 0.8988 ± 0.0544 | 0.8543 ± 0.0288 |
| 0.2 | Uni | 0.9841 ± 0.0178 | 0.9154 ± 0.0217 | 0.9219 ± 0.0333 | 0.8454 ± 0.0283 |
|  | UAPN | 0.9868 ± 0.0181 | 0.9293 ± 0.0306 | 0.9152 ± 0.0229 | 0.8712 ± 0.0266 |
|  | LAPN | 0.9645 ± 0.0189 | 0.9310 ± 0.0227 | 0.9030 ± 0.0503 | 0.8516 ± 0.0400 |
| 0.3 | Uni | 0.9777 ± 0.0264 | 0.9257 ± 0.0288 | 0.8975 ± 0.0454 | 0.8506 ± 0.0310 |
|  | UAPN | 0.9868 ± 0.0125 | 0.9309 ± 0.0355 | 0.9346 ± 0.0395 | 0.8500 ± 0.0363 |
|  | LAPN | 0.9745 ± 0.0174 | 0.9352 ± 0.0266 | 0.9172 ± 0.0373 | 0.8536 ± 0.0343 |
| 0.4 | Uni | 0.9782 ± 0.0200 | 0.9278 ± 0.0295 | 0.9112 ± 0.0327 | 0.8377 ± 0.0292 |
|  | UAPN | 0.9815 ± 0.0178 | 0.9160 ± 0.0464 | 0.8988 ± 0.0419 | 0.8509 ± 0.0350 |
|  | LAPN | 0.9718 ± 0.0271 | 0.9325 ± 0.0401 | 0.9243 ± 0.0512 | 0.8499 ± 0.0650 |
| 0.5 | Uni | 0.9742 ± 0.0183 | 0.8858 ± 0.1505 | 0.9341 ± 0.0350 | 0.7787 ± 0.2008 |
|  | UAPN | 0.9692 ± 0.0197 | 0.8587 ± 0.2278 | 0.9082 ± 0.0758 | 0.7914 ± 0.2033 |
|  | LAPN | 0.9725 ± 0.0184 | 0.9006 ± 0.0635 | 0.9129 ± 0.0467 | 0.8357 ± 0.0334 |

Table 11. $\Omega_{\alpha}$ Results as a function of knowledge-distillation hyper-parameter $\lambda_2$, and ‘loss-aware weighted class balancing replacement’ (LAWCBR) and different sampling strategies for (i) i.i.d, (ii) class-i.i.d, (iii) instance, and (iv) class-instance ordering on iCubWorld 1.0 dataset.

- **streaming class-i.i.d**: We follow the similar approach as has been mentioned for CIFAR100 streaming class-i.i.d ordering.

**G.4. iCubWorld 1.0**

iCubWorld 1.0 [9] is an object recognition dataset containing the sequence of video frames, with each frame containing only a single object. It is a more challenging and realistic dataset w.r.t the other standard datasets such as CIFAR10, CIFAR100, and ImageNet100. Technically, it is an ideal dataset to evaluate a model’s performance in streaming learning scenarios that are known to induce catastrophic forgetting [12], as it requires learning from temporally ordered image sequences, which are naturally non-i.i.d images.
| $\lambda_2$ | Sample Selection | iCubWorld 1.0 |
|-----------|-----------------|--------------|
|           |                 | iid          | Class-iid | Instance | Class-instance |
| 0.0       | Uni             | 0.9298 ± 0.0329 | 0.9063 ± 0.0396 | 0.8837 ± 0.0544 | 0.9168 ± 0.0312 |
|           | UAPN            | 0.9184 ± 0.0379 | 0.8818 ± 0.0396 | 0.7507 ± 0.0732 | 0.8384 ± 0.0675 |
|           | LAPN            | 0.9285 ± 0.0357 | 0.8912 ± 0.0430 | 0.7735 ± 0.0458 | 0.8657 ± 0.0521 |
| 0.1       | Uni             | 0.9830 ± 0.0207 | 0.9240 ± 0.0276 | 0.9292 ± 0.0344 | 0.9162 ± 0.0255 |
|           | UAPN            | 0.9644 ± 0.0260 | 0.9368 ± 0.0228 | 0.9439 ± 0.0362 | 0.9411 ± 0.0224 |
|           | LAPN            | 0.9541 ± 0.0280 | 0.9402 ± 0.0368 | 0.9241 ± 0.0401 | 0.9345 ± 0.0235 |
| 0.2       | Uni             | 0.9600 ± 0.0312 | 0.9351 ± 0.0315 | 0.9155 ± 0.0299 | 0.9229 ± 0.0284 |
|           | UAPN            | 0.9640 ± 0.0236 | 0.9415 ± 0.0307 | 0.9254 ± 0.0331 | 0.9468 ± 0.0273 |
|           | LAPN            | 0.9684 ± 0.0160 | 0.9382 ± 0.0361 | 0.9368 ± 0.0376 | 0.9454 ± 0.0263 |
| 0.3       | Uni             | 0.9716 ± 0.0141 | 0.9118 ± 0.0344 | 0.9269 ± 0.0383 | 0.9346 ± 0.0191 |
|           | UAPN            | 0.9454 ± 0.0239 | 0.9480 ± 0.0215 | 0.9580 ± 0.0298 | 0.9585 ± 0.0223 |
|           | LAPN            | 0.9667 ± 0.0174 | 0.9538 ± 0.0303 | 0.9558 ± 0.0304 | 0.9497 ± 0.0239 |
| 0.4       | Uni             | 0.9611 ± 0.0153 | 0.9243 ± 0.0524 | 0.9350 ± 0.0319 | 0.9222 ± 0.0403 |
|           | UAPN            | 0.9647 ± 0.0257 | 0.9387 ± 0.0315 | 0.9476 ± 0.0264 | 0.9005 ± 0.1257 |
|           | LAPN            | 0.9615 ± 0.0194 | 0.9323 ± 0.0421 | 0.9257 ± 0.0212 | 0.9509 ± 0.0323 |
| 0.5       | Uni             | 0.9615 ± 0.0301 | 0.9391 ± 0.0268 | 0.9001 ± 0.0555 | 0.9145 ± 0.0649 |
|           | UAPN            | 0.9526 ± 0.0179 | 0.9390 ± 0.0267 | 0.9275 ± 0.0322 | 0.8766 ± 0.2320 |
|           | LAPN            | 0.9495 ± 0.0215 | 0.9085 ± 0.1230 | 0.9369 ± 0.0187 | 0.9533 ± 0.0248 |

Table 12. $\Omega_{i.i.d}$ Results as a function of knowledge-distillation hyper-parameter $\lambda_2$, and ‘loss-aware weighted random replacement with a reservoir’ (LAWRRR) and different sampling strategies for (i) i.i.d, (ii) class-i.i.d, (iii) instance, and (iv) class-instance ordering on iCubWorld 1.0 dataset.

| Parameters | CIFAR10 | CIFAR100 | ImageNet100 | iCubWorld 1.0 |
|------------|---------|----------|-------------|--------------|
| Optimizer  | SGD     | SGD      | SGD         | SGD          |
| Learning Rate | 0.01  | 0.01    | 0.01        | 0.01         |
| Momentum   | 0.9     | 0.9     | 0.9         | 0.9          |
| Weight Decay | 1e-05 | 1e-05   | 1e-05       | 1e-05        |
| Hidden Layer | [256, 256] | [256, 256] | [256, 256] | [256, 256] |
| Activation | ReLU    | ReLU     | ReLU        | ReLU         |
| Offline Batch Size | 128   | 128     | 256         | 16           |
| Offline Epoch | 50    | 50      | 100         | 30           |
| Buffer Capacity | 1000  | 1000    | 1000        | 180          |
| Train-Set Size | 50000 | 50000   | 127778      | 6002         |

Table 13. Training parameters used for CIOSL and Offline model.

It contains 10 classes, each with 3 different object instances with 200 – 201 images each. Overall, each class contains 600 – 602 samples for training and 200 – 201 samples for testing. Figure 6 shows example images of the 30 object instances in iCubWorld 1.0, where each row denotes one of the 10 categories.

We use iCubWorld 1.0 to evaluate the performance of the streaming learner’s in all the four data ordering schemes, i.e., (i) streaming i.i.d, (ii) streaming class-i.i.d, (iii) streaming instance, and (iv) streaming class-instance.

- **streaming i.i.d:** In this setting, we follow the similar approach as mentioned for the CIFAR10 dataset, with the only exception, that is, 10% randomly selected samples are used for the base initialization, and the rest are used for streaming learning.

- **streaming class-i.i.d:** In this case, we follow the same strategy as mentioned for CIFAR10 streaming class-i.i.d ordering.

- **streaming class-instance:** In base initialization, the model is trained in a typical offline mode with the samples from the first two classes. In each incremental step,
the network is trained in a streaming manner with the samples from the succeeding two classes which were not observed earlier. However, in this case, (i) samples within a class are temporally ordered based on different object instances, and (ii) all samples from one class are fed into the network before feeding any samples from the other class.

- **streaming instance**: For the base initialization, 10% randomly chosen samples are used, and the remaining samples are used to train the model incrementally with one sample at a time. In streaming setting, the samples are temporally ordered based on different object instances. Specifically, we organize the data stream by putting temporally ordered 50 frames of an object instance, then we put temporally ordered 50 frames of the second object instance, and so on. In this way, after putting 50 temporally ordered frames from each object instance, we put the next 50 temporally ordered frames of the first object instance and follow the earlier approach until all the frames of each instance have been exhausted.
H. Derivation of Joint Posterior

\[ L'_t(\theta) = \arg \min_{q_t \in \mathcal{Q}} KL \left[ q_t(\theta) \mid\mid \frac{1}{z_t} q_{t-1}(\theta)p(D_t|\theta)p(D_{M,t}|\theta) \right] \]

\[ \simeq \arg \min_{q_t \in \mathcal{Q}} KL \left[ q_t(\theta) \mid\mid q_{t-1}(\theta)p(D_t|\theta)p(D_{M,t}|\theta) \right] \]

\[ = \int q_t(\theta) \log \frac{q_t(\theta)}{q_{t-1}(\theta)p(D_t|\theta)p(D_{M,t}|\theta)} d\theta \]

\[ = - \int q_t(\theta) \log \frac{1}{q_{t-1}(\theta)p(D_t|\theta)p(D_{M,t}|\theta)} \frac{q_t(\theta)}{q_t(\theta)} d\theta \]

\[ = \arg \max \int q_t(\theta) \log \frac{q_{t-1}(\theta)p(D_t|\theta)p(D_{M,t}|\theta)}{q_t(\theta)} d\theta \]

\[ = \int q_t(\theta) \log p(D_t|\theta) d\theta + \int q_t(\theta) \log p(D_{M,t}|\theta) d\theta \]

\[ - \int q_t(\theta) \log \frac{q_t(\theta)}{q_{t-1}(\theta)} d\theta \]

\[ = \mathbb{E}_{\theta \sim q_t(\theta)} \left[ \log p(D_t|\theta) \right] + \mathbb{E}_{\theta \sim q_t(\theta)} \left[ \log p(D_{M,t}|\theta) \right] \]

\[ - \lambda_1 \cdot KL \left[ q_t(\theta) \mid\mid q_{t-1}(\theta) \right] \]

\[ = \arg \max \mathbb{E}_{\theta \sim q_t(\theta)} \left[ \log p(y_t|\theta, G(x_t)) \right] \]

\[ + \sum_{n=1}^{N'_t} \mathbb{E}_{\theta \sim q_t(\theta)} \left[ \log p(y^{(n)}_{M,t}|\theta, z^{(n)}_{M,t}) \right] \]

\[ - \lambda_1 \cdot KL \left( q_t(\theta) \mid\mid q_{t-1}(\theta) \right) \]

where \( D_t = \{d_t\} = \{(x_t, y_t)\} \), \( D_{M,t} = \{d^{(n)}_{M,t}\}_{n=1}^{N'_t} = \{(z^{(n)}_{M,t}, y^{(n)}_{M,t})\}_{n=1}^{N'_t} \), \( |D_{M,t}| = N'_t \ll |M| \), \( D_{M,t} \subset M \), and \( \lambda_1 \) is a hyper-parameter.