Single-Net Continual Learning with Progressive Segmented Training

Xiaocong Du  
School of ECEE  
Arizona State University  
Tempe, AZ, USA  
xiaocong@asu.edu

Gouranga Charan  
School of ECEE  
Arizona State University  
Tempe, AZ, USA  
gcharan@asu.edu

Frank Liu  
CSMD  
Oak Ridge National Lab  
Oak Ridge, TN, USA  
luiufy@ornl.gov

Yu Cao  
School of ECEE  
Arizona State University  
Tempe, AZ, USA  
ycao@asu.edu

Abstract—There is an increasing need of continual learning in dynamic systems, such as the self-driving vehicle, the surveillance drone, and the robotic system. Such a system requires learning from the data stream, training the model to preserve previous information and adapt to a new task, and generating a single-headed vector for future inference. Different from previous approaches with dynamic structures, this work focuses on a single network and model segmentation to prevent catastrophic forgetting. Leveraging the redundant capacity of a single network, model parameters for each task are separated into two groups: one important group which is frozen to preserve current knowledge, and secondary group to be saved (not pruned) for a future learning. A fixed-size memory containing a small amount of previously seen data is further adopted to assist the training. Without additional regularization, the simple yet effective approach of Progressive Segmented Training (PST) successfully incorporates multiple tasks and achieves the state-of-the-art accuracy in the single-head evaluation on CIFAR-10 and CIFAR-100 datasets. Moreover, the segmented training significantly improves computation efficiency in continual learning at the edge.

Index Terms—Continual learning, convolutional neural network, deep learning, computer vision, image recognition

I. INTRODUCTION

The rapid advancement of computing and sensing technology has enabled many new applications, such as the self-driving vehicle, the surveillance drone, and the robotic system. Compared to conventional edge devices (e.g., cell phone or smart home devices), these emerging devices are required to deal with much more complicated, dynamic situations. One of the necessary attributes is the capability of continual learning: when encountering a sequence of tasks over time, the learning system should capture the new observation and update its knowledge (i.e. the network parameters) in real time, without interfering or overwriting previously acquired knowledge. Recent literature [3]–[10] have intensively studied this topic, and it is believed that, in order to learn a data stream continually, such a system should have the following features:

Online adaption. The system should be able to update its knowledge according to a continuum of data, without independent and identically distributed (i.i.d.) assumption on this data stream. For a dynamic system (e.g. a self-driving vehicle), it is preferred that such adaption is completed locally and in real time.

Preservation of prior knowledge. When new data arrives in a stream, previous data are very limited or even no longer exists. Yet the acquired knowledge from previous data should not be forgotten (i.e. overwritten or deteriorated due to the learning of new data). In other words, the prior distribution of the model parameters should be preserved.

Single-head evaluation. The network should be able to differentiate the tasks and achieve successful inter-task classification without the prior knowledge of the task identifier (i.e. which task current data belongs to). In the case of single-head, the network output should consist of all the classes seen so far. In contrast, multi-head evaluation only deals with intra-task classification where the network output only consists of a subset of all the classes. Multi-head classification is more appropriate for multi-task learning than continual learning [3].

Resource constraint. The resource usage such as the model size, the computation cost, and storage requirements should be bounded during continual learning from sequential tasks, rather than increasing proportionally or even exponentially over time.

For the aforementioned features, one of the serious challenges is catastrophic forgetting of the prior knowledge. McCloskey et al. [11] first identified the catastrophic forgetting problem in the connectionist networks. Henceforth, various solutions to mitigate catastrophic forgetting have been proposed. These solutions can be categorized into two families: (1) Dynamic network structure. These methods [9], [10], [12]–[15] usually expand the new knowledge by growing the network structure. For example, [12] progressively adds new network branches for new tasks and keeps previously learned features in lateral connections. In this case, prior knowledge and new knowledge are usually separated into different feed-forward paths. Moreover, the newly added branches have never been exposed to the previous data and thus is blind to previous tasks. Due to these fundamental reasons, the performance of dynamic architectures on the single-head classification lags behind, although they were able to maintain the accuracy in multi-head classification with the priori of task identification. The second family is (2) single network structure. In contrast...
to a dynamic structure, these methods learn sequential tasks with a single, static network structure all the time. The knowledge of prior and new tasks are packed in a single network that is exposed to all tasks over time. In this case, the challenge is shifted to minimizing the interference among tasks and preserving prior knowledge in the same network. As a contemporary neural network has a large capacity to accommodate multiple tasks, we believe a single network provides a promising basis for continual learning.

In the family of the single-network methods, previous works have explored the regularization methods [2], [3], [16]–[18], the parameter isolation methods [19], [20] and the memory rehearsal methods [5], [7], [8], [21], [22]. The regularization methods leverage a penalty term in the loss function to regularize the parameters when updating for new tasks. However, as more and more tasks appear, the parameters tend to be biased towards the new tasks, and the system gradually drifts away from previous distribution. To mitigate such a knowledge asymmetry, regularization methods can be combined with memory rehearsal methods [23], [24]. Recent works such as iCaRL [8] and GEM [7] have proven the efficacy of replaying the memory (i.e., maintain the system with a subset of the previously seen data) in abating the network parameters drifting far away from previous knowledge. Parameter isolation approaches [19], [20] allocate subsets of parameters for previous tasks and prune the rest for learning new tasks. In this case, the rest of the parameters no longer contain prior knowledge, violating the aforementioned properties of an ideal continual learning system. For instance, PackNet [19] and Piggyback [20] achieve strong performance on multi-head evaluation but not on single-head.

To achieve continual learning with the preservation of prior knowledge, we propose single-net continual learning with Progressive Segmented Training, namely PST, as shown in Fig. 1 When new data comes in, PST adapts the network parameters with memory-assisted balancing, then important parameters are identified according to their contribution to this task. Next, to alleviate catastrophic forgetting, PST performs model segmentation by reinforcing important parameters (through retraining) and then freezing them in the future training; while the secondary parameters will be saved (not pruned) and updated by future tasks. Through experiments on CIFAR-10 [25] and CIFAR-100 [25] dataset with modern deep neural networks, we demonstrate that PST achieves state-of-the-art single-head accuracy and successfully preserves the previously acquired knowledge in the scenario of continual learning. Moreover, benefiting from model segmentation, the amount of computation needed to learn a new task keeps reducing. This property brings PST high efficiency in computation as compared to other regularization methods.

The contribution of this paper is as follows:

• We summarize important features of a successful continual learning system and propose a novel training scheme, namely Progressive Segmented Training (PST), to mitigate catastrophic forgetting in continual learning.

• Different from previous works in which new observation overwrites the entire acquired knowledge, PST leverages parameter segmentation for each task to prevent knowledge overwriting or deterioration. Experiments on CIFAR-10 and CIFAR-100 dataset prove that PST successfully alleviates catastrophic forgetting and reaches state-of-the-art single-head accuracy in the learning of streamed data, with $24\times$ reduction in computation cost as compared to the previous work.

• We demonstrate a detailed ablation study and discussion to analyze the role of each component in PST.

II. METHOD

In this section, we first describe the terminology and algorithm of PST. Then we interpret three major components: memory-assisted training and balancing, importance sampling and model segmentation in Section II-B Section II-C and Section II-D respectively.

A. Overview of PST

a) Terminology: The continual learning problem can be formulated as follows: the machine learning system is continuously exposed to a stream of labeled input data $X^1, X^2, \ldots$, where $X^y = \{x^1, \ldots, x^y\}$ correspond to all examples of class $y \in \mathbb{N}$. When the new task $\{X^s, \ldots, X^t\}$ comes in, the data of old tasks $\{X^1, \ldots, X^{s-1}\}$ are no longer available, except a small amount of previously seen data stored in the memory set $P = \{P_1, \ldots, P_{s-1}\}$.

For a modern deep neural network such as VGG-Net [26] and ResNet [27], the network parameter $\Theta$ usually consists of feature extractor $\varphi : \mathcal{X} \to \mathbb{R}^d$ and classification weight vectors $w \in \mathbb{R}^d$. The network keeps updating its parameter $\Theta$ according to the previously seen data $\mathcal{X}$, in order to predict labels $y^*$ with its output $\mathcal{Y} = w^\top \varphi(\mathcal{X})$. During training the network with data corresponding to classes $\{X^1, \ldots, X^{s-1}\}$, our target is to minimize the loss function $\mathcal{L}(Y; \mathcal{X}^s; \Theta)$ of this $(s-1)$-class classifier. Similarly, with the introduction of a new task with classes $\{X^s, \ldots, X^t\}$, the target now is to minimize $\mathcal{L}(Y; \mathcal{X}^t; \Theta)$ of this $t$-class classifier.

Algorithm 1 PST training routine
Input:$\{X^s, \ldots, X^t\}$
Require $\Theta = (\Theta_{\text{fixed}}; \Theta_{\text{free}})$
Require $\mathcal{P} = (P_1, \ldots, P_{s-1})$
1: Memory-assisted training and balancing: $\Theta_{\text{free}} \to \Theta_{\text{free}}'$
2: Importance sampling: identify $\Theta_{\text{important}}'$ in $\Theta_{\text{free}}'$
3: Model segmentation: $\Theta_{\text{important}}' \to \Theta_{\text{important}}''$
4: $(\Theta_{\text{fixed}}; \Theta_{\text{important}}'') \to \Theta_{\text{fixed}}''$
5: $\Theta_{\text{secondary}} \to \Theta_{\text{free}}'$
Output: $\Theta' = (\Theta_{\text{fixed}}''; \Theta_{\text{free}}')$
Output: $\mathcal{P}' = (P_1, \ldots, P_t)$
b) Training routine: Every time when a new task is available, PST calls a training routine (Fig. 1 and Algorithm 1) to update the parameter \( \Theta \) to \( \Theta' \), and the memory set \( \mathcal{P} \) to \( \mathcal{P}' \), according to the current training data \( \{X^1, \ldots, X^n\} \) and a small amount of previously seen data (memory set) \( \mathcal{P} \). The training routine consists of three major components: (1) memory-assisted training and balancing, (2) importance sampling and (3) model segmentation, as illustrated in the following subsections.

B. Memory-assisted training and balancing

Fig. 1 illustrates PST training routine for task \( T_i \) and task \( T_{i+1} \). In Fig. 1, which is the moment that task \( T_i \) comes in, the network consists of two portions: parameters \( \Theta_{\text{fixed}} \) (grey blocks) are fixed for previous tasks, and parameters \( \Theta_{\text{free}} \) (light blue blocks) are trainable for current and future tasks. We allow \( \Theta_{\text{free}} \) to be updated for task \( T_i \), with \( \Theta_{\text{fixed}} \) included in the feedforward path. To mitigate the parameters bias towards new task, a memory set is used to assist the training. The memory set is sampled uniformly and randomly from all the classes in previous tasks, which is a simple yet highly efficient approach, as explained in RWalk work [1]. For example, if the memory budget is \( K \) and \( s-1 \) classes have been learned in previous tasks, then the memory set stores \( \frac{K}{s-1} \) images for each class. We mix samples from this memory set with equal samples per class from the current task, i.e. \( K \) samples of the memory and \( \frac{K}{s-1} \times (t-s+1) \) samples from current task, and provide them to the network: (i) for a few epochs at the beginning of the training; (ii) periodically (e.g. every 3 epochs) during training; (iii) for a few epochs at the end of the training to fine-tune classification layer (i, ii, iii are noted in Fig. 3). In comparison to most related works, which adopt the single-stage (step ii) optimization technique, the proposed three-step optimization strategy performs much better. One of the primary reasons behind catastrophic forgetting is knowledge drift in both feature extraction and classification layers. The three-pronged strategy helps minimize this drift in the following ways: step i provides a well-balanced initialization; step ii reviews previous data and thus, consolidates previous learned knowledge for the entire network; step iii corrects bias by balancing classification layers, which is simple yet efficient as compared to [28] that utilizes an extra Bias Correction Layer after the classifier. After memory-assisted training and balancing, the network parameters are updated from \( \hat{\Theta} = (\Theta_{\text{fixed}}; \Theta_{\text{free}}) \) to \( \hat{\Theta}' = (\Theta_{\text{fixed}}; \Theta_{\text{free}}') \), as stated in Algorithm 1 line 1.

C. Importance sampling

After the network has learned on task \( T_i \), PST samples crucial learning units for the current task: for feature extraction layers (i.e. convolutional layers), PST samples important filters; for fully-connected layers, PST samples important neurons. The definitions of filter and neuron are as follows:

The \( l \)-th convolutional layer can be formulated as: the output of this layer \( Y_l = X_l \ast \Theta_l \), where \( \Theta_l \in \mathbb{R}^{O_l \times I_l \times K \times K} \). The set of weights that generates the \( o \)-th output feature map is denoted as a filter \( \Theta^o_l \), where \( \Theta^o_l \in \mathbb{R}^{I_l \times K \times K} \). The \( l \)-th fully-connected layer can be represented by: \( Y_l = X_l \ast \Theta_l \), where \( \Theta_l \in \mathbb{R}^{O_l \times I_l} \). The set of weights \( \Theta^o_l \) that connected to the \( t \)-th class can be denoted as a neuron, where \( \Theta^o_l \in \mathbb{R}^{1 \times I_l} \).

The filter/neuron sampling is based on an importance score that is adopted in PST to measure the effect of a single filter/neuron on the loss function, i.e. the importance of each filter/neuron. The importance score is developed from the Taylor Expansion of the loss function. Previously, Molchanov et al. [29] applied it on pruning secondary parameters. The importance score represents the difference between the loss with and without each filter/neuron. In other words, if the removal of a filter/neuron leads to relatively small accuracy degradation, this unit is recognized as an unimportant unit, and vice versa. Thus, the objective function to get the filter with the highest importance score is formulated as:

\[
\arg\min_{\Theta^o_l} |\Delta \mathcal{L}(\Theta^o_l)| \iff \arg\min_{\Theta^o_l} |\mathcal{L}(Y; X; \Theta) - \mathcal{L}(Y; X; \Theta^m_l) = 0) |
\]  

(1)
Using the first-order of Taylor Expansion of $|\mathcal{L}(Y;X;\Theta) - \mathcal{L}(Y;X;\Theta^0)|$ at $\Theta^0 = 0$, we get:

$$|\Delta \mathcal{L}(\Theta^0)| \approx \left| \frac{\partial \mathcal{L}(Y;X;\Theta^0)}{\partial \Theta^0} \right| = \sum_{i=0}^{K} \sum_{m=0}^{K} \sum_{n=0}^{K} \left| \frac{\partial \mathcal{L}(Y;X;\Theta^0)}{\partial \Theta^0} \right| \Theta^{i,m,n}$$

(2)

where $\frac{\partial \mathcal{L}(Y;X;\Theta^0)}{\partial \Theta^0}$ is the gradient of the loss function with respect to parameter $\Theta^{i,m,n}$.

Similarly, the saliency score of a neuron is derived as:

$$|\Delta \mathcal{L}(\Theta^t)| \approx \left| \frac{\partial \mathcal{L}(Y;X;\Theta^t)}{\partial \Theta^t} \right| = \sum_{i=0}^{K} \sum_{m=0}^{K} \sum_{n=0}^{K} \left| \frac{\partial \mathcal{L}(Y;X;\Theta^t)}{\partial \Theta^t} \right| \Theta^{i,t,m,n}$$

(3)

where $\frac{\partial \mathcal{L}(Y;X;\Theta^t)}{\partial \Theta^t}$ is the gradient of the loss with respect to parameter $\Theta^{i,t,m,n}$.

Based on the importance score, we sort the learning units layer by layer and identify the top $\beta$ units (dark blue blocks in Fig. 1(b)). In the following model segmentation step, we deal with the location of important parameters, rather than the value of these parameters, which will be explained in the next subsection. $\beta$ is an empirical hyper-parameter that should be approximately proportional to the complexity of the current task. For example, when incrementally learning 10 classes of CIFAR-100 at a time, $\beta$ can be 10%; when learning 20 classes per task, $\beta$ can be 20%.

Due to the nature of continual learning, the total number of tasks is not known beforehand, so the network can be reserved with a larger capacity in order to freeze enough knowledge for previous tasks and leave enough space for future tasks. Once the continual learning is complete, one can leverage model compression approaches [30]–[34] to compress the model size.

It is also worth mentioning that importance sampling is only performed once after each task, so that the computation cost of this step is negligible.

D. Model segmentation and reinforcement

After important units are sampled according to the importance score, current network parameter $\Theta' = (\Theta_{fixed}; \Theta_{important}; \Theta_{secondary})$, where $\Theta_{fixed}$ are the frozen parameters for all the previous tasks, $\Theta_{important}$ are important parameters for the current task, and $\Theta_{secondary}$ are unimportant parameters for the current task, as stated in Algorithm 1 line 2. Our ideal target is to reinforce $\Theta_{important}$ in a way such that their contribution to the current task is as crucial as possible. Previously, Liu et al. [35] observed that the sampled network architecture itself (rather than the selected parameters) is more indispensable to the learning efficacy. Inspired by this conclusion, we keep the $\Theta_{fixed}$ and $\Theta_{secondary}$ intact, randomly initialize $\Theta_{important}$ and retrain them with current training data assisted by memory set to obtain $\Theta_{important}$. This step reinforces the contribution of $\Theta_{important}$ to the learning, as proved by our experimental results demonstrated in Fig. 2 and Table I. After model segmentation, $\Theta_{important}$ along with the aforementioned $\Theta_{fixed}$ will be kept frozen in the future tasks, and $\Theta_{secondary}$ will be used to learn new knowledge.

III. EXPERIMENTAL RESULTS

In this section, we present experimental results to verify the efficacy of PST. The experiments are performed with PyTorch [36] on one NVIDIA GeForce RTX 2080 platform.

Datasets. The CIFAR [25] dataset consists of 50,000 training images and 10,000 testing images in color with size $32 \times 32$. There are 10 classes for CIFAR-10 and 100 classes for CIFAR-100. In Section III-A, CIFAR-10 is divided into 2 tasks, i.e., 5 classes per task, to provide a comprehensive analysis of PST. In Section III-B and III-C following iCaRL [8], CIFAR-100 is divided into 5, 10, 20 or 50 classes per task, to demonstrate extensive experiments. For each experiment, we shuffle the class order and run 5 times to report the average accuracy.

Network structures. In the following experiment, the structure and size of VGG-16 [26] we use for CIFAR-10 dataset follows [26]. The structure and size of 32-layer ResNet for CIFAR-100 dataset follows the design of iCaRL [8]. Each convolutional layer in VGG-16 and ResNet is followed by a batch normalization layer [37]. As aforementioned in Section II-C, the number of classes will occur is unknown in a continual learning scenario. Thus, we leave 1.2× space at the final classification layer in the following experiments, i.e., 12 outputs for CIFAR-10 and 120 outputs for CIFAR-100. It is worth mentioning that the number of classes reserved at the final classification layer does not affect the overall performance, as there is no feedback from vacant classes.

Experimental setup. Standard Stochastic Gradient Descent with momentum 0.9 and weight decay 5E-4 are used for training. The initial learning rate is set to 0.1 and is divided by 10 for every 40% and 80% of the total training epochs. On CIFAR-10 and CIFAR-100 datasets, we train 180 and 100 epochs at the stage of memory-assisted training and balancing, 120 and 60 epochs at the stage of model segmentation. $\beta$ is set as proportional to the complexity of the current task. For example, when incrementally learning 5 classes of CIFAR-10 at a time, $\beta$ is set to 50%; when incrementally learning 10 classes of CIFAR-100 at a time, $\beta$ is set to 10%. The memory storage is set as $K = 2000$ images for a fair comparison with the previous work [8].

Evaluation protocol. As mentioned in Section I, single-head evaluation is more practical and valuable than multi-head evaluation in the scenario of continual learning. Therefore, we evaluate single-head accuracy for the following experiments.

To report the single-head overall accuracy if input data $\{X^t\}$ have been observed so far, we test the network with testing data that sampled uniformly and randomly from class 1 to class $t$ and predict a label out of $t$ classes $\{1, \ldots, t\}$. For the first task accuracy (such as Fig 5), we test the network with testing data collected from the first task $T_1$ (supposing classes $\{1, \ldots, g\}$) and predict a label out of $t$ classes $\{1, \ldots, t\}$ to report single-head $T_1$ accuracy (Fig. 5(a)); or, predict a label out of $g$ classes $\{1, \ldots, g\}$ to report multi-head $T_1$ accuracy (Fig. 5(b)).
### A. In-depth analysis

We divide CIFAR-10 into 2 tasks (5 classes each) and analyze the PST training routine step by step in this subsection. The learning curve is present in Fig. 3.

From epoch 0 to epoch 180, $T_1$ is trained and reaches baseline accuracy. The weight distribution after training $T_1$ is present in Fig. 2a. At epoch 180, we sample the top 50% (since there are 2 tasks totally) important parameters and retrain them with the secondary parameters untouched (epoch 180 to epoch 300), which is the model segmentation step. The weight distribution after this step is shown in Fig. 2d. It is worth mentioning that previous works, such as PackNet [20] and Piggyback [19], prune the secondary parameters and thus, distort the weight distribution (Fig. 2b). At epoch 300, task $T_2$ appears and updates the parameters.

At the same time, the acquired knowledge of $T_1$ is disturbed by $T_2$ updating, leading to an accuracy degradation on $T_1$ (see the green curve at epoch 300). From epoch 300 to the end is the step of $T_2$ training, during which the memory data is injected following (i), (ii), and (iii) to balance. After $T_2$ training, we again plot the weight distribution for the pruning-based approach (in Fig. 2c) and PST approach (in Fig. 2e). It is observed that the pruning approach fails to preserve the prior knowledge, as the weight distribution after learning $T_2$ shifts far away from the previous one. In contrast, PST well preserves prior knowledge (i.e. similar weight distribution after learning $T_1$ and after learning $T_2$). Compared to the baseline accuracy, pruning-based approaches forget 31% on overall accuracy while segmentation-based PST only forgets 5%.

### B. Extensive results

#### Accuracy for incrementally learning multi-classes.

We compare PST with state-of-the-art approaches that reported single-head accuracy: MAS [16], EWC [1], RWalk [4], SI [2], LwF.MC [18], DMC [10], iCaRL.MC [8] and two baselines: fixed representation, finetuning. Fixed representation denotes the method that we fix the feature extraction layers for the previously learned tasks and only train classification layers for new tasks. Finetuning denotes the method that the network trained on previous tasks is directly fine-tuned by new tasks, without strategies to prevent catastrophic forgetting. LwF.MC denotes the method that uses LwF [18] but is evaluated with multi-class single-head classification. iCaRL.MC denotes the method uses iCaRL but replaces their Nearest-Mean-of-Exemplar [8] classifier with a regular output classifier for a fair comparison with PST. The results of MAS, EWC, RWalk, SI and DMC are from [10], which is implemented with the official code [1]. The results of fix representation, finetune, LwF.MC, and iCaRL are from [8]. We adopt the same memory size for fair comparison between baselines and PST.

The single-head overall accuracy when incrementally learning 20 tasks (5 classes per task), 10 tasks (10 classes per task), 5 tasks (20 classes per task) and 2 tasks (50 classes per task) are reported in Fig. 4. Among 9 different approaches, PST achieves the best accuracy on the 2-task scenario and the second best accuracy on the other scenarios. Compare to [1]https://github.com/facebookresearch/agem
Fig. 4. Single-head overall accuracy on CIFAR-100 when incrementally learning 20, 10, 5, 2 tasks in a sequence. PST has the best accuracy of 2 tasks and the second best accuracy of 5, 10, 20 tasks. Though iCaRL.MC has better accuracy than PST, it requires \( >24\times \) computation cost than PST (see Fig. 6 for details).

(a) Single-head accuracy

(b) Multi-head accuracy

Fig. 5. (a) Single-head accuracy and (b) multi-head accuracy of the first task \( T_1 \) over time when the model is trained with a sequence of 20 tasks on CIFAR-100.

finetuning, PST largely prevents the model from catastrophic forgetting. Though PST achieves lower accuracy than iCaRL in some cases, PST is more than \( 24\times \) efficient in computation cost, as shown in Fig. 6. This efficiency is benefiting from model segmentation: iCaRL has to update the entire network parameters for every new observation, but PST only requires updating partial network parameters, as the parameters related to previous tasks are frozen. Meanwhile, PST outperforms iCaRL in the multi-head protocol, as present in the following paragraph.

Accuracy of the first task. Fig. 5(a) compares the single-head accuracy on the first task \( T_1 \) in PST with several previous approaches that reported \( T_1 \) accuracy in their papers. It also presents the multi-head accuracy on \( T_1 \) in PST and the baseline accuracy. PST achieves the best single-head accuracy on \( T_1 \) among all the approaches, i.e. the least forgetting. Moreover, when \( T_1 \) data is evaluated in a multi-head classification setting, as shown in Fig. 5(b) PST is stable and always on par with the baseline (the model that is only trained on \( T_1 \), so without forgetting). This phenomenon demonstrates that PST effectively preserves the knowledge related to \( T_1 \) through model segmentation. Without these strategies, it is hard to keep the previously acquired knowledge. For example, GEM reported unstable multi-head \( T_1 \) accuracy, which is because the parameters gradually drift away from \( T_1 \) knowledge after a long period of learning on new tasks.

C. Efficient edge learning

In a more realistic situation (such as self-driving cars), continual learning may not be used to train a model from scratch at the edge. Instead, we will have a model which is well trained in the cloud (i.e. knowledge repository) and once deployed, might only be required to learn a few new classes in an online manner on the edge devices. For example, 90 classes of CIFAR-100 are pre-trained in the cloud and 10 classes are expected to be learned on edge devices continually. Moreover, computation cost is a critical overhead when deploying Deep
Neural Networks on edge devices [30], [31], [38], [39]. Therefore, edge learning prefers algorithms with low computation cost rather than that with higher one.

For such a scenario, we estimate the training computation cost, i.e., the number of floating point operations (FLOPs), required by PST and regularization approaches such as iCaRL [8] and EWC [1], as shown in Fig. 6. Training on hardware includes three paths [40], i.e., (1) forward path, (2) backward path, and (3) weight update path. As more and more tasks come in, the trainable parameters become fewer and fewer in PST, i.e., weight update path gradually requires fewer operations, but regularization methods require a constant number of operations at all times, as the model is not segmented. Thus, given the model is pre-trained in the cloud with a large amount of data (e.g., 90 classes) and loaded at the edge to learn new observations (e.g., 10 classes), PST reduces more than 24× FLOPs in the weight update path, and more than 1.5× FLOPs in complete path (including all three paths), as compared to the regularization methods such as iCaRL [8]. Especially, weight update path usually costs 2× latency than the other two paths so that PST can largely speed up the training. Benefiting from segmentation, PST outperforms other continual learning schemes in computation efficiency.

IV. ABLATION STUDY AND DISCUSSION

In this section, we analyze the importance of each component in PST by performing an ablation study and demonstrate that each component in PST is contributing to the overall performance.

Fig. 7 illustrates the effect of different memory budgets. With more data saved from previous tasks, the forgetting is reduced. But such a trend gradually saturates. Considering resource constraint, the learning system cannot rely on memory rehearsal itself to recover catastrophic forgetting.

To analyze the role of each component in PST, we switch off each component in PST and repeat the experiments performed in Fig. 4. Replacing importance sampling with a random sampling leads to model Hybrid 1; removing model segmentation step (no reinforcement on Θ_i) leads to model Hybrid 2; removing the memory-assisted balancing leads to model Hybrid 3. The overall accuracy change after the last task is reported in Table I. The results of hybrid models prove that importance sampling and model segmentation are indispensable steps for PST since removing them leads to significant accuracy drop, and memory-assisted balancing is supplementary.

V. CONCLUSION

A successful continual learning system that is exposed to a continuous data stream should have the properties of online adaption, preservation of prior knowledge, single-head evaluation and resource constraint, to alleviate or even prevent catastrophic forgetting of previously acquired knowledge. To satisfy these properties and minimize catastrophic forgetting, we propose a novel scheme named single-net continual learning with Progressive Segmented Training (PST). Benefiting from its components (memory-assisted training and balancing, importance sampling, and model segmentation), PST achieves state-of-the-art single-head accuracy on incremental tasks on CIFAR datasets, with far lower computation cost. We further demonstrate that PST favors edge computing due to its segmented training method. In future work, we plan to study the detailed mechanism of catastrophic forgetting further and improve PST. Moreover, we plan to explore compressing or even eliminating the memory data without sacrificing the performance.

ACKNOWLEDGMENT

This work was supported in part by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. It is also partially support by National Science Foundation (CCF #1715443).

REFERENCES

[1] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska et al., “Overcoming catastrophic forgetting in neural networks,” Proceedings of the national academy of sciences, vol. 114, no. 13, pp. 3521–3526, 2017.
[2] F. Zenke, B. Poole, and S. Ganguli, “Continual Learning Through Synaptic Intelligence,” arXiv:1703.04200 [cs, q-bio, stat], Mar. 2017, arXiv: 1703.04200. [Online]. Available: http://arxiv.org/abs/1703.04200
[3] R. Aljundi, M. Lin, B. Goujaud, and Y. Bengio, “Online continual learning with no task boundaries,” arXiv:1903.08671 [cs, stat], Mar. 2019, arXiv: 1903.08671. [Online]. Available: http://arxiv.org/abs/1903.08671
[4] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, “Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence,” arXiv:1801.10112 [cs], vol. 11215, pp. 556–572, 2018, arXiv: 1801.10112. [Online]. Available: http://arxiv.org/abs/1801.10112
[5] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, “EFFICIENT LIFELONG LEARNING WITH A-GEM,” p. 20, 2019.
[6] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, “Overcoming catastrophic forgetting by incremental moment matching,” in Advances in neural information processing systems, 2017, pp. 4652–4662.
K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

Y. Wu, Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu, “Large scale incremental learning,” CoRR, vol. abs/1905.13260, 2019. [Online]. Available: http://arxiv.org/abs/1905.13260

P. Molchanov, S. Tyree, I. Kuriia, I. Anil, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” arXiv preprint arXiv:1611.06440, 2016.

S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” arXiv preprint arXiv:1608.08710, 2016.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” The Journal of Machine Learning Research, vol. 18, no. 1, pp. 6869–6898, 2017.

X. Du, Z. Li, Y. Ma, and Y. Cao, “Efficient network construction through structural plasticity,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 3, pp. 453–464, Sep. 2019.

Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” arXiv preprint arXiv:1412.6115, 2014.

Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” arXiv preprint arXiv:1810.05270, 2018.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.

S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” arXiv preprint arXiv:1502.03167, 2015.

J.-H. Luo, J. Wu, and W. Lin, “Thinet: A filter level pruning method for deep neural network compression,” in Proceedings of the IEEE international conference on computer vision, 2017, pp. 5058–5066.

X. Du, G. Krishnan, A. Mohanty, Z. Li, G. Charan, and Y. Cao, “Towards efficient neural networks on-a-chip: Joint hardware-algorithm approaches,” in 2019 China Semiconductor Technology International Conference (CSTIC), March 2019, pp. 1–5.

S. Kolala Venkatakrishnan, Y. Ma, S. Yin, E. Nurvitadhi, A. Dasu, Y. Cao, and J.-s. Seo, “Automatic compiler based fpga accelerator for cnn training,” in 2019 29th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2019.