Efficient continual learning at the edge with progressive segmented training

Xiaocong Du, Shreyas Kolala Venkataramanaiah, Zheng Li, Han-Sok Suh, Shihui Yin, Gokul Krishnan, Frank Liu, Jae-sun Seo, and Yu Cao

1 School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287, United States of America
2 School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85287, United States of America
3 Computer Science and Mathematics Division, Oak Ridge National Lab, Oak Ridge, TN 37830, United States of America

* Author to whom any correspondence should be addressed.
E-mail: xiaocong@asu.edu and ycao@asu.edu

Keywords: continual learning, acquisitive learning, deep neural network, brain inspiration, model adaption

Abstract

There is an increasing need for continual learning in dynamic systems at the edge, such as self-driving vehicles, surveillance drones, and robotic systems. 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, within a limited power budget. Different from previous continual learning algorithms with dynamic structures, this work focuses on a single network and model segmentation to mitigate catastrophic forgetting problem. 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 a secondary group to be saved (not pruned) for 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 state-of-the-art accuracy in the single-head evaluation on the CIFAR-10 and CIFAR-100 datasets. Moreover, the segmented training significantly improves computation efficiency in continual learning and thus, enabling efficient continual learning at the edge. On Intel Stratix-10 MX FPGA, we further demonstrate the efficiency of PST with representative CNNs trained on CIFAR-10.

1. Introduction

The rapid advancement of computing and sensing technology has enabled many new edge applications, such as self-driving vehicle, surveillance drone, and robotic systems. Compared to conventional edge devices (e.g. cell phones or smart home devices), these emerging devices are required to deal with much more complicated and dynamic situations with limited power budgets. One of the necessary attributes is the capability of efficient continual learning (i.e. online learning): when encountering a sequence of tasks over time, the edge device should capture the new observation and update its knowledge (i.e. the network parameters) in real-time, without interfering or overwriting previously acquired knowledge, and such learning should be computationally efficient at the edge. Recent literature [3–10] have intensively studied this topic. It is believed that to achieve efficient online learning, such an edge computing 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.) assumptions 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 exist. 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 system 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 neural 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. Due to the limited power and memory budget at the edge, 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. McClosey 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 feedforward 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 prior task identification.

(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 is 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, 4, 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 toward the new tasks, and the system gradually drifts away from the 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. train 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 figure 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 throughout the future training procedure; while the secondary parameters will be saved (not pruned) and updated by the future training tasks. Through experiments on CIFAR-10 [25] and CIFAR-100 [25] datasets 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 makes PST to exhibit high efficiency in computation as compared to other regularization methods. We prove the efficiency of PST with both simulated results and real-time demonstration on the FPGA platform.

The key contributions of this paper are as follows:

- We summarize important features of a successful continual learning system and propose a novel training scheme, namely 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.
- We prove the effectiveness of PST on the CIFAR-10 and CIFAR-100 dataset, showing that PST successfully alleviates catastrophic forgetting and reaches state-of-the-art single-head accuracy in the learning of streamed data.
• We present the advantage of PST in the scenario of edge computing from the perspective of accuracy and computation cost. With the FPGA-based 16 bit fixed-point training accelerator, we further validate that PST significantly reduces computational costs when learning at the edge.

The rest of this paper is organized as follows. Section 2 describes previous efforts on continual learning. Section 3 presents the training routine of PST as well as a detailed description of each component. Section 4 demonstrates an in-depth analysis of PST on CIFAR-10 and extensive results on CIFAR-100 when learning streamed tasks. Section 5 emphasizes the efficiency of PST when learning at the edge, and validate by simulated results and an FPGA demo. Section 6 presents the ablation study of each component in PST and memory budget. Finally, we conclude this work in section 7.

2. Related work

In this section, we review previous efforts to alleviate catastrophic forgetting in continual learning. Prior works can be largely divided into two categories: (1) dynamic network structure and (2) single network structure.

Dynamic network structure. Methods with expandable or growing network structures are categorized in this family. [12] progressively adds a new branch of neural networks for each new task and leaves the old knowledge untouched. [14] expands a fixed amount of neurons to learn new knowledge and partially retrains weights that are associated with old tasks. However, in both methods, the newly added branches or neurons have never been trained on old input data, limiting the model performance on the entire dataset. [9] combines two individual models that are trained on old and new classes through dual distillation. [13] uses reinforcement learning to adaptively expand each layer of the network when a new task arrives. Due to the nature of dynamic structures, the inference of old and new tasks are separated in different paths, and thus these methods usually perform better on the multi-head protocol. Compared to the dynamic network family, the proposed PST encodes the entire knowledge of all the tasks into a single network in order to achieve single-head evaluation.

Single network structure. In contrast to the dynamic network, some previous works embody all the tasks in a single network, i.e. static network structure. 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. Techniques such as regularization, parameter isolation, and memory rehearsal (including pseudo memory) are explored.

Regularization. To constrain the learning between new and old classes, some prior works [1, 2, 16] add a penalty term in the objective function to regularize the parameter updating for new tasks, or use knowledge distillation [18, 26, 27] and bias correction [27]. Along with learning more and more tasks, network parameters gradually drift away and become biased toward new tasks since regularization is a soft constraint on parameter updating. Different from them, PST does not require an additional term in loss function and applies hard constraint on parameter updating rather than soft constraint.

Parameter isolation. PackNet [20] iteratively prunes unimportant weights and fine-tunes them in the learning of new tasks. Similarly, Piggyback [19] prunes network parameters by learning a mask from network quantization. PackNet [20] and Piggyback [19] achieve strong performance on multi-head evaluation but not on single-head. We argue that pruning secondary parameters is sub-optimal in the case of single-head protocol.
since pruning destroys parameter distribution. Detailed discussion is provided in section 4.1. Different from PackNet and Piggyback, PST implements segmentation by consolidating important parameters for past tasks and saving secondary parameters for new tasks. In other words, new tasks are learned from scratch (weights are zero) in PackNet and Piggyback, and thus old tasks and new tasks are disjoint. In PST, however, new tasks are learned based on old tasks so that weight distribution can be preserved.

**Memory rehearsal and pseudo memory rehearsal.** To mitigate knowledge bias toward new tasks, some methods store previous data and retrain them [5, 7, 8, 21, 22], or train generative adversarial networks (GANs) to generate and discriminate images and then learn the data distribution [28–31]. Memory rehearsal methods require additional storage to store previous data or extra model parameters to generate and discriminate data. However, scalability is not a concern as long as the storage or the GAN model size is constrained in the learning of streamed data.

### 3. Method

In this section, we first describe the terminology and algorithm of PST. Then we interpret three major components: memory-assisted training and balancing, significance sampling, and model segmentation in sections 3.2–3.4, respectively.

#### 3.1. Overview of PST

**Terminology.** The continual learning problem can be formulated as follows: the machine learning system is continuously exposed to a stream of labeled input data \( X = \{X^t, X^2, \ldots \} \), where \( X^t = \{x^t_1, \ldots, x^t_N\} \) correspond to all examples of class \( y \in \mathbb{N} \). When the new task \( \{X^t, \ldots, X^{t+1}\} \) comes in, the data of old tasks \( \{X^1, \ldots, X^{t-1}\} \) are no longer available, except for a small amount of previously seen data stored in the memory set \( P = (P_1, \ldots, P_{t-1}) \).

For deep neural networks such as VGG-Net [32] and ResNet [34], the network parameter \( \Theta \) usually consists of feature extractor \( \varphi : \mathcal{X} \rightarrow \mathbb{R}^d \) and classification weight vectors \( w \in \mathbb{R}^d \). The network keeps updating its parameter \( \Theta \) according to the previously seen data \( X^t \), in order to predict labels \( Y^t \) with its output \( \mathcal{Y} = \varphi(X) \). During the network training with data corresponding to classes \( \{X^t, \ldots, X^{t-1}\} \), our target is to minimize the loss function \( \mathcal{L}(Y; X^t; \Theta) \) of this \( (s-1) \)-class classifier. Similarly, with the introduction of a new task with classes \( \{X^t, \ldots, X^{t+1}\} \), the target now is to minimize \( \mathcal{L}(Y; X^t; \Theta) \) of this \( t \)-class classifier.

**Training routine.** Every time when a new task is available, PST calls a training routine (figure 1 and algorithm 1) to update the parameter \( \Theta \) to \( \Theta' \), and the memory set \( P \) to \( P' \), according to the current training data \( \{X^t, \ldots, X^{t+1}\} \) and a small amount of previously seen data (memory set) \( P \). The training routine consists of three major components: (1) memory-assisted training and balancing, (2) significance sampling, and (3) model segmentation, as described in the following subsections.

#### 3.2. Memory-assisted training and balancing

Figure 1 illustrates PST training routine for task \( T_i \) and task \( T_{i+1} \). In figure 1(a), which is the moment that task \( T_i \) comes in, the network consists of two portions: parameters \( \Theta_{\text{fixed}} \) (gray 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 parameter bias toward the 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 the RWalk work [4]. 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 the 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–iii are noted in figure 3).

In comparison to most related works that adopt the single-stage 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 previously learned knowledge for the entire network; step (iii) corrects bias by balancing classification layers, which is simple yet efficient as compared to [27] that utilizes an extra bias correction layer after the classifier. After memory-assisted training and balancing, the network parameters are updated from $\Theta = (\Theta_{\text{fixed}}; \Theta_{\text{important}}; \Theta_{\text{secondary}})$ to $\Theta' = (\Theta'_{\text{fixed}}; \Theta'_{\text{important}}; \Theta'_{\text{secondary}})$, as stated in algorithm 1 line 1.

3.3. Significance sampling

After the network has learned 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 $Y_l = X_l \cdot \Theta_l$, where $\Theta_l \in \mathbb{R}^{O_l \times K \times K}$. The set of weights that generates the $l$th output feature map is denoted as a filter $\Theta^f_l$, where $\Theta^f_l \in \mathbb{R}^{K \times K}$. The $l$th fully-connected layer can be represented by: $Y_l = X_l \cdot \Theta_l$, where $\Theta_l \in \mathbb{R}^{O_l \times K}$. The set of weights $\Theta^f_l$ that connected to the $l$th class can be denoted as a neuron, where $\Theta^f_l \in \mathbb{R}^{1 \times K}$.

The filter/neuron sampling is based on an importance score that is adopted in PST to measure the effect of each 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 [35] 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 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 obtain the filter with the highest importance score is formulated as:

$$\arg\min_{\Theta^f_l} |\Delta L(\Theta^f_l)| \Leftrightarrow \arg\min_{\Theta^f_l} |L(Y_l; X_l; \Theta_l) - L(Y_l; X_l; \Theta^f_l = 0)| \quad (1)$$

Using the first-order of Taylor expansion of $|L(Y_l; X_l; \Theta_l) - L(Y_l; X_l; \Theta^f_l = 0)|$ at $\Theta^f_l = 0$, we get:

$$|\Delta L(\Theta^f_l)| \simeq |\frac{\partial L(Y_l; X_l; \Theta_l)}{\partial \Theta^f_l} \cdot \Theta^f_l| = \sum_{i=0}^h \sum_{m=0}^K \sum_{n=0}^K \frac{\partial L(Y_l; X_l; \Theta_l)}{\partial \Theta^f_{l,m,n}} \Theta^f_{l,m,n} \quad (2)$$

where $\frac{\partial L(Y_l; X_l; \Theta_l)}{\partial \Theta^f_{l,m,n}}$ is the gradient of the loss function with respect to parameter $\Theta^f_{l,m,n}$.

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

$$|\Delta L(\Theta^f_l)| \simeq |\frac{\partial L(Y_l; X_l; \Theta_l)}{\partial \Theta^f_l} \cdot \Theta^f_l| = \sum_{i=0}^h |\frac{\partial L(Y_l; X_l; \Theta_l)}{\partial \Theta^f_{l,i}} \cdot \Theta^f_{l,i}| \quad (3)$$

where $\frac{\partial L(Y_l; X_l; \Theta_l)}{\partial \Theta^f_{l,i}}$ is the gradient of the loss with respect to parameter $\Theta^f_{l,i}$.

Based on the importance score, we sort the learning units layer by layer and identify the top $\beta$ units (dark blue blocks in figure 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 [36–40] to compress the model size. It is also worth mentioning that significance sampling is only performed once after each task so that the computation cost of this step is minimized.

3.4. Model segmentation and reinforcement

After important units are sampled according to the importance score, current network parameter $\Theta' = (\Theta'_{\text{fixed}}; \Theta'_{\text{important}}; \Theta'_{\text{secondary}})$, where $\Theta'_{\text{fixed}}$ are the frozen parameters for all the previous tasks, $\Theta'_{\text{important}}$ are important parameters for the current task, and $\Theta'_{\text{secondary}}$ are unimportant parameters for the current task, as stated in algorithm 1 line 2. Our ideal target is to reinforce $\Theta'_{\text{important}}$ in a way such that their contribution to the current task is as crucial as possible. Previously, Liu et al [41] 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 $\Theta_{\text{fixed}}$ and $\Theta_{\text{secondary}}$ intact, randomly initialize $\Theta_{\text{important}}$ and retrain them with current training data assisted by a memory set to obtain $\Theta_{\text{important}}'$. This step reinforces the contribution of $\Theta_{\text{important}}$ to the learning, as proved by our experimental results demonstrated in figure 2 and table 2. After model segmentation, $\Theta_{\text{important}}'$ along with the aforementioned $\Theta_{\text{fixed}}$ will be kept frozen in future tasks, and $\Theta_{\text{secondary}}$ will be used to learn new knowledge.

4. Accuracy: learning streamed tasks

In this section, we present experimental results to verify the efficacy of PST. The experiments are performed with PyTorch [42] 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 4.1, CIFAR-10 is divided into 2 tasks, i.e. 5 classes per task, to provide a comprehensive analysis of PST. In section 4.2, 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 [32] we use follow [32]. The structure and size of 32-layer ResNet follow the design of iCaRL [8]. Each convolution layer in VGG-16 and ResNet is followed by a batch normalization layer [43]. As mentioned in section 3.3, the number of new classes that will be learned is unknown in a continual learning scenario. Thus, we leave $1.2 \times$ 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 (SGD) with the momentum of 0.9 and weight decay of $5 \times 10^{-4}$ is 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 the CIFAR-10 and CIFAR-100 datasets, we train 180 and 100 epochs at the stage of memory-assisted training and balancing, and 120 and 60 epochs at the stage of model segmentation. The memory storage is set as $K = 2000$ images for a fair comparison with the previous work [8].

Evaluation protocol. As mentioned in section 1, 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 $\{X_1, \ldots, X_t\}$ 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 figure 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 (figure 5(a)); or, predict a label out of $g$ classes $\{1, \ldots, g\}$ to report multi-head $T_1$ accuracy (figure 5(b)).

4.1. 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. Figure 3 presents the learning curve for training 2 tasks (5 classes each) in CIFAR-10.

From epoch 0 to epoch 180, $T_1$ is trained and reaches baseline accuracy. The weight distribution after training $T_1$ is present in figure 2(a). At epoch 180, we sample the top 50% (since there are two tasks in total) 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 figure 2(d). It is worth...
mentioning that previous works, such as PackNet [20] and Piggyback [19], prune the secondary parameters and thus, distort the weight distribution (figure 2(b)). 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 step (i)–(iii) to balance.

After $T_2$ training, we again plot the weight distribution for the pruning-based approach (in figure 2(c)) and PST approach (in figure 2(e)). 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%.

4.2. Extensive results

On the CIFAR-100 dataset, our experimental results show that: (1) in overall accuracy, PST outperforms most of the previous work [1, 2, 4, 10, 16, 18] and is on par with iCaRL [8]. (2) With model segmentation, PST successfully preserves prior knowledge. (3) PST reduces more than $24 \times$ computation cost in edge computing, as compared to classic regularization approaches.

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 fixes the feature extraction layers for the previously learned tasks and only trains 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. The results of fix representation, finetune, LwF.MC and iCaRL are from [8]. We adopt the same memory size for a fair comparison between the 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 figure 4. Among 9 different approaches, PST achieves the best accuracy on the 2-task scenario and the second best accuracy on the other scenarios. Compared to finetuning, PST largely prevents the model from catastrophic forgetting. Although PST achieves lower accuracy than iCaRL in some cases, PST is more than $24 \times$ efficient in the computation cost, as shown in figure 6. This efficiency is benefiting from model segmentation: iCaRL has to update the entire network parameters for every new observation, but PST only requires the update of partial network parameters, as the parameters related to previous tasks are frozen.

Accuracy of the first task. Figure 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. 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 figure 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 difficult to

4 https://github.com/facebookresearch/agem
Figure 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 figure 6 for details).

Figure 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.

5. Computation cost: learning at the edge

5.1. Simulated results

In a more realistic situation, 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 and once deployed, might only be required to learn a few new classes in an online manner on the edge devices. In this section, we developed experiments to show that PST benefits continual training at the edge from the perspective of accuracy and computation cost.

In table 1, we test such a system where the base model is pre-trained (similar to training on the cloud) with 10, 30, 50, 70, or 90 classes of CIFAR-100 as task $T_1$, while the new task $T_2$ that consists of 10 disjoint classes has to be learned at the edge continually. The number of trainable parameters for $T_2$ remains the same across these 5 experiments. As shown in table 1, if large amounts of data have been well trained in the cloud and stored in the segmented PST model, the training of incremental data at the edge causes marginal forgetting (e.g. 0.08) of the acquired knowledge.
Table 1. With increasing data trained in the cloud, PST effectively mitigates forgetting. Note that in this experiment, the network size is much smaller than that in section 4.2.

| Classes (T₁ + T₂) | Accuracy (after T₁) | Accuracy' (after T₂) | Forgetting (ΔAccuracy') |
|-------------------|---------------------|----------------------|-------------------------|
| 10 + 10           | 0.77                | 0.32                 | 0.45                    |
| 30 + 10           | 0.78                | 0.60                 | 0.18                    |
| 50 + 10           | 0.78                | 0.64                 | 0.14                    |
| 70 + 10           | 0.79                | 0.67                 | 0.12                    |
| 90 + 10           | 0.77                | 0.69                 | 0.08                    |

ΔAccuracy = Accuracy − Accuracy'.

Moreover, we estimate the computation cost during training, 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 figure 6. Computation cost is a critical overhead when deploying deep neural networks on edge devices [36, 37, 39, 44]. Edge learning prefers algorithms with low computation cost rather than that with higher one. Training at the edge includes three paths [45, 46], 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., the 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 and loaded at the edge, PST reduces the FLOPs in the weight update path by more than 24×, and by more than 1.5× in the complete path (including all three paths), as compared to the regularization methods such as iCaRL [8]. Especially, the weight update path usually consumes 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.

5.2. FPGA demonstration

In this section, we demonstrate online CIFAR-10 CNN learning on an FPGA-based 16 bit fixed-point training accelerator [45, 47] on Intel Stratix-10 MX FPGA [48]. The CNN training hardware is flexible to support forward pass (FP), backward pass (BP) and weight update (WU) phases of training.

Figure 7 presents the overall FPGA system setup [49] to train CNNs using PST algorithm. For simplicity, the CNN structure used here is 16C3-16C3-MP-32C3-32C3-MP-64C3-64C3-MP-FC, where ‘NCk’ refers to the convolution layer with ‘N’ output feature maps and a kernel size of ‘k’, ‘MP’ refers to max-pooling layer and ‘FC’ refers to fully-connected layer. First, as shown in figure 7(a), a large amount of weights is pre-trained...
and selected with 9 classes from CIFAR-10 dataset. The pre-trained model and a binary mask representing the frozen weights are fed to the RTL generator. The RTL generator generates the customized training accelerator based on the pre-trained model structure and generates HBM2 memory initialization files to load the model parameters, as shown in figure 7(b). The frozen weights stored in HBM2 are used by the FPGA training accelerator to perform inference on pre-trained classes.

The model is then exposed to a new, unlearned class from the CIFAR-10 dataset, and updated accordingly in real-time on the FPGA, as shown in figure 7(c). The entire system is demonstrated on Intel Stratix-10 MX FPGA board (figure 7(d)). Benefiting from the model segmentation, the online training of new observations requires much less computation cost and lower latency, as compared to the traditional continual learning scheme that updates the entire network. As shown in figure 7(e), the breakdown graph shows that the PST scheme saves $4.2 \times$ latency per image in the weight update (WU) phase as compared to traditional algorithms.

6. Ablation study and discussion

In this section, we analyze the importance of each component in PST by performing an ablation study and demonstrate that PST is highly efficient in edge computing by virtue of single-net segmentation.

6.1. Analysis of each component in PST

We remove each component from PST and repeat the experiments performed in figure 4. The overall accuracy change after the last task is reported in table 2. Replacing significance sampling with random sampling leads to model Hybrid 1; removing the model segmentation step (no reinforcement on $\Theta_{\text{important}}$) leads to model Hybrid 2; removing the memory-assisted balancing leads to model Hybrid 3. The results of hybrid models prove that each component in PST is contributing to the overall performance. Removing significance sampling or model segmentation leads to a significant accuracy drop, while removing memory leads to a small accuracy drop. It shows that significance sampling and model segmentation are indispensable steps for PST, and memory-assisted balancing is supplementary.

6.2. Memory budget

For PST, the accuracy gap between single-head and multi-head of $T_1$ could be caused by the imbalance between old and new knowledge (the network is biased to new knowledge than old knowledge since old data are no longer used to train the network). Memory-assisted balancing in PST alleviates this obstacle but cannot completely prevent it. Indeed, there has hitherto been no approach to prevent this knowledge asymmetry. With more data saved from previous tasks, forgetting is reduced. But such a trend gradually saturates, as shown in figure 8.
Table 2. Switching off different components of PST leads to accuracy drop to different extents. In the table, negative numbers indicate an accuracy drop, e.g. $-0.32$ means 32% accuracy drop.

| Model                                           | 20 tasks | 10 tasks | 5 tasks |
|-------------------------------------------------|----------|----------|---------|
| Hybrid 1 (removing significance sampling)       | $-0.32$  | $-0.38$  | $-0.45$ |
| Hybrid 2 (removing model segmentation)          | $-0.32$  | $-0.38$  | $-0.42$ |
| Hybrid 3 (removing memory balancing)            | $-0.06$  | $-0.08$  | $-0.11$ |

Figure 8. Overall single-head accuracy when incrementally learning 10 tasks under different memory budget.

7. Conclusion

A successful continual learning system that is exposed to a continuous data stream should exhibit 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 PST. Benefiting from memory-assisted training and balancing, significance sampling, and model segmentation, PST outperforms the state-of-the-art single-head accuracy ($+16\%$) and multi-head accuracy ($+15\%$) on incremental tasks on the CIFAR-100 dataset, with significantly 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 performance.

Acknowledgments

This work was partially supported by C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

ORCID iDs

Xiaocong Du https://orcid.org/0000-0002-1079-0347

References

[1] Kirkpatrick J et al 2017 Proc. Natl Acad. Sci. USA 114 3521–6
[2] Zenke F, Poole B and Ganguli S 2017 arXiv:1703.04200 [cs, q-bio, stat]
[3] Aljundi R, Lin M, Goujard B and Bengio Y 2019 arXiv:1903.08671 [cs, stat]
[4] Chaudhry A, Dokania P K, Ajanthan T and Torr P H S 2018 arXiv:1801.10112 [cs]
[5] Chaudhry A, Ranzato M, Rohrbach M and Elhoseiny M 2018 arXiv:1812.00420
[6] Lee S W, Kim J H, Jun J, Ha J W and Zhang B T 2017 Overcoming catastrophic forgetting by incremental moment matching Advances in Neural Information Processing Systems pp 4652–62
[7] Lopez-Paz D and Ranzato M 2017 Gradient episodic memory for continual learning Advances in Neural Information Processing Systems vol 30 ed I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan and R Garnett (Curran Associates, Inc.) pp 6467–76 http://papers.nips.cc/paper/7225-gradient-episodic-memory-for-continual-learning.pdf
