CONTINUAL LEARNING VIA LOW-RANK NETWORK UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Continual learning seeks to train a single network for multiple tasks (one after another), where training data for each task is only available during the training of that task. Neural networks tend to forget older tasks when they are trained for the newer tasks; this property is often known as catastrophic forgetting. To address this issue, continual learning methods use episodic memory, parameter regularization, masking and pruning, or extensible network structures. In this paper, we propose a new continual learning framework based on low-rank factorization. In particular, we represent the network weights for each layer as a linear combination of several low-rank (or rank-1) matrices. To update the network for a new task, we learn a low-rank (or rank-1) matrix and add that to the weights of every layer. We also introduce an additional selector vector that assigns different weights to the low-rank matrices learned for the previous tasks. We show that our approach performs better than the current state-of-the-art methods in terms of accuracy and forgetting. Our method also offers better memory efficiency compared to episodic memory-based approaches.

1 INTRODUCTION

Deep neural networks have been extremely successful for a variety of learning and representation tasks (e.g., image classification, object detection/segmentation, reinforcement learning, generative models). A typical network is trained to learn a function that maps input to the desired output. The input-output relation is assumed to be fixed and input-output data samples are drawn from a stationary distribution Parisi et al. (2019). If the input-output relations or data distributions change, the network can be retrained using a new set of input-output data samples. Since the storage, computing, and network capacity are limited, we may need to replace old data samples with new samples. Furthermore, privacy concerns may also force data samples to be available for a limited time Delange et al. (2021); Parisi et al. (2019). In such a training process, a network often forgets the previously learned tasks; this effect is termed *catastrophic forgetting* McCloskey & Cohen (1989); Ratcliff (1990).

Continual learning or *lifelong learning* approaches aim to address the problem of catastrophic forgetting by adapting the network or training process to learn new tasks without forgetting the previously learned ones Nguyen et al. (2018); Li & Hoiem (2017); Aljundi et al. (2017; 2018; 2019); Chaudhry et al. (2018); Riemer et al. (2019); Rolnick et al. (2019); Farajtabar et al. (2020). In this paper, we focus on task-incremental continual learning in which data for every task are provided in a sequential manner to train/update the network Chaudhry et al. (2019a). Let us denote the network function that maps input x to output for task t as $f(x; \mathcal{W}_t)$, where $\mathcal{W}_t$ denotes the network weights for task t. We seek to update the $\mathcal{W}_t$ for all t as we sequentially receive dataset for one task at a time. Suppose the training dataset for task t is given as $(\mathcal{X}_t, \mathcal{Y}_t)$ drawn from a distribution $\mathcal{P}_t$, where $\mathcal{X}_t$ denotes the set of input samples and $\mathcal{Y}_t$ denotes the corresponding ground-truth outputs. Our goal is to update network weights form the previous task ($\mathcal{W}_{t-1}$) to $\mathcal{W}_t$ such that

$$y \approx f(x; \mathcal{W}_t), \quad \text{for all } (x, y) \sim \mathcal{P}_t. \tag{1}$$

The setup above assumes that the task identity of test samples is known at the test time and the corresponding network weights are used for inference. This is in contrast to a more challenging case where we may need to identify the task along with the label Wortsman et al. (2020). Furthermore,

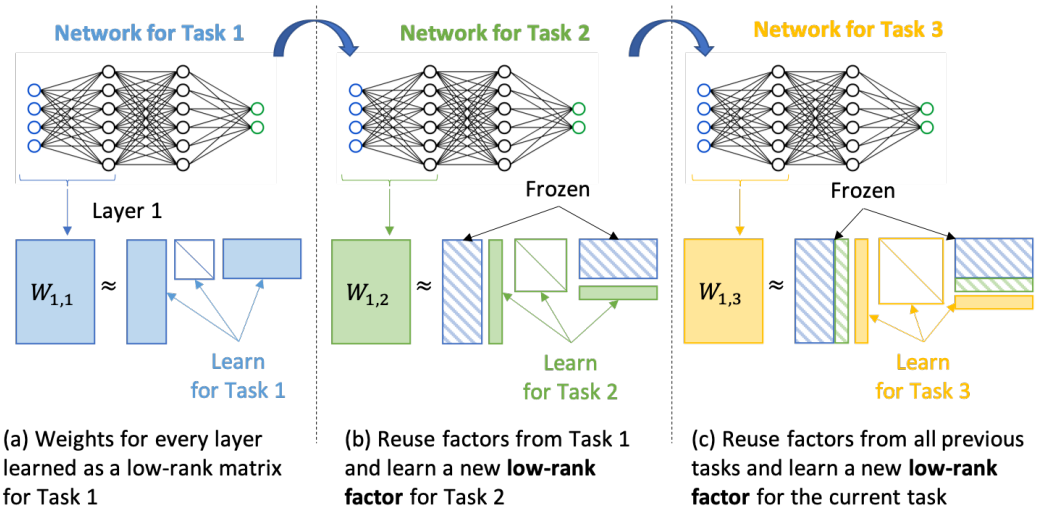

Figure 1: An overview of our proposed method for continual learning via low-rank network updates. We first represent (and learn) the weight matrix (or tensor) for each layer as a product of low-rank matrices. To train a network for new tasks without forgetting the earlier tasks, we reuse the factors from the earlier tasks and add a new set of factors for the new task. Our experiments suggest that a rank-one update is often sufficient for successful continual learning.

using $\mathcal{W}_t$ for testing data for task t every time ensures *zero forgetting*; however, this also requires storing the $\mathcal{W}_t$ for all the tasks. One of the main contributions of this paper is to represent, learn, and update the $\mathcal{W}_t$ using low-rank factors such that they can be stored and applied with minimal memory and computation overhead.

We propose a new method for continual learning that updates network weights using rank-one (or low-rank) increments. Figure 1 provides an illustration of our proposed method. We represent the network weights for each layer as a linear combination of several low-rank factors (which can be represented as a product of two low-rank matrices and a diagonal matrix). To update the network for task t without forgetting the earlier tasks, we freeze the low-rank factors learned from the previous tasks, add a new trainable rank-1 (or low-rank) factor for every layer, and combine that with the older factors using learnable *selector weights* (shown as a diagonal matrix). We use a multi-head configuration that has an independent output layer for each task. We present an extensive set of experiments to demonstrate the performance of our proposed method for different benchmark datasets. We observe that our proposed method outperforms the current state-of-the-art methods in terms of accuracy with small memory overhead.

The main contributions of this paper are as follows.

1. **Represent layers as low-rank matrices:** We represent and learn network weights for each layer as a low-rank structure. We show that low-rank structure is sufficient to represent all the tasks in continual learning setup.
2. **Reuse old factors for better performance with a small memory overhead:** We limit the number of parameters required for network update by reusing the factors learned from previous tasks. We demonstrate that a rank-1 increment suffices to outperform the existing techniques.
3. **Zero forgetting without replay buffer:** Our method has zero forgetting that is achieved using incremental rank update or network weights. In contrast, most of the existing continual learning techniques require replay buffer or large memory overhead to achieve zero forgetting.

Limitations. Our approach has a few limitations. Since we use all the previously learned factors for inference, the later tasks require more memory and computation for inference. Nevertheless, we show that using low-rank structure, our total memory requirement is much lower than a single network. Our method also requires the knowledge of task identity at the time of inference.

2 BACKGROUND AND RELATED WORK

Continual learning Delange et al. (2021) or lifelong learning Silver & Mercer (2002) aims to train a single model on a sequence of different tasks and perform well on all the trained tasks once the training is finished. While training on new tasks, the old data from previous tasks will not be provided to the model. This scenario mimics the human learning process where they have the ability to acquire new knowledge and skills throughout their lifespan. However, this setting is still challenging to neural network models as a common phenomenon called "catastrophic forgetting McCloskey & Cohen (1989)" is observed during this learning process, where the data from the new tasks interfere with the data seen in the previous tasks and thus deteriorating model performance on preceding tasks. To overcome this issue, different approaches have been proposed so far which can be divided into three main categories: regularization-based approaches, memory and replay-based approaches, and dynamic network architecture-based approaches.

Regularization-based approaches Kirkpatrick et al. (2017); Nguyen et al. (2018); Li & Hoiem (2017) update the whole model in each task but a regularization term ℓ_{reg} is added to the total loss $\mathcal{L} = \ell_{current} + \lambda \ell_{reg}$ to penalize changes in the parameters important to preceding tasks thus preserving the performance on previous learned tasks. For example, Elastic Weight Consolidation (EWC) Kirkpatrick et al. (2017) estimates the importance of parameters using Fisher Information matrix; Variational Continual Learning (VCL) Nguyen et al. (2018) approximates the posterior distribution of the parameters using variational inference; Learning without Forgetting (LwF) Li & Hoiem (2017) regularizes the current loss with soft targets taken from previous tasks using knowledge distillation Hinton et al. (2014).

Memory-based approaches Rebuffi et al. (2017); Riemer et al. (2019); Chaudhry et al. (2019a;b); Tang et al. (2021) usually use memory and replay mechanism to recall a small episodic memory of previous tasks while training new tasks thus reduce the loss in the previous tasks. For example, iCaRL Rebuffi et al. (2017) is the first replay method, which learns in a class-incremental way by selecting and storing exemplars closest to the feature mean of each class; Meta-Experience Replay (MER) Riemer et al. (2019) combines experience replay with optimization-based meta-learning to optimize the symmetric trade-off between transfer and interference by enforcing gradient alignment across examples; AGEM Chaudhry et al. (2019a) projects the gradient on the current minibatch by using an external episodic memory of patterns from previous experiences as an optimization constraint; ER-Ring Chaudhry et al. (2019b) jointly trains new task data with that of the previous tasks.

Dynamic network architecture Rusu et al. (2016); Mallya & Lazebnik (2018); Wortsman et al. (2020); Wen et al. (2020); Serra et al. (2018); Chaudhry et al. (2020); Yoon et al. (2018) try to add new neurons to the model at additional new tasks, thus the performances on previous tasks are preserved by freezing the old parameters and only updating the newly added parameters. For example, Progressive neural networks (PNNs) Rusu et al. (2016) leverage prior knowledge via lateral connections to previously learned features; PackNet Mallya & Lazebnik (2018) iteratively assigns parameter subsets to consecutive tasks by constituting binary masks. SupSup Wortsman et al. (2020) also finds masks in order to assign different subsets of the weights for different tasks. BatchEnsemble Wen et al. (2020) learns on separate rank-1 scaling matrices for each task which are then used to scale weights of the shared network. HAT Serra et al. (2018) incorporates task-specific embeddings for attention masking. ORTHOG-SUBSPACE Chaudhry et al. (2020) learn tasks in different (low-rank) vector sub-spaces that are kept orthogonal to each other in order to minimize interference.

Our proposed method falls under the category of dynamic network architecture approaches. Note that we can represent a low-rank weight matrix using two smaller fully-connected layers and increasing the rank of the weight matrix is equivalent to adding new nodes in the two smaller fully-connected layers.

3 CONTINUAL LEARNING VIA RANK INCREMENT

We focus on the continual learning setup in which we seek to train a network for T tasks. The main difference between continual learning and regular learning is that the training data for every task is only available while training the network for that task. The main challenge in continual learning is to not forget the previous tasks as we learn new tasks. Learning each task entails training weights for

the network to learn the task-specific input-output relationship using the task-specific training data. We represent the training data for task t as $(\mathcal{X}_t, \mathcal{Y}_t)$ drawn from distribution $\mathcal{P}_t$. We represent the network function for task t as $f(x; \mathcal{W}_t)$, where $\mathcal{W}_t$ denote the network weights we have to learn for task t. We seek to develop a continual learning framework in which we represent the weights of any layer using a small number of low-rank factors. We initialize the network with a base architecture in which weights for each layer can be represented using a low-rank matrix. We then add new low-rank factors to each layer as we learn new tasks.

Let us assume the network has K layers and the weights for the kth layer and task t can be represented as $W_{k,t}$. Let us further assume that the weights for the kth layer and task $t = 1$ can be represented as a low-rank matrix

$$W_{k,1} = U_{k,1} S_{k,1,1} V_{k,1}^\top, \tag{2}$$

where $U_{k,1}, V_{k,1}$ represent two low-rank matrices and $S_{k,1,1}$ represents a diagonal matrix. To learn the network for task 1, we learn $U_{k,1}, V_{k,1}, S_{k,1,1}$ for all k. For task 2, we represent the weights for kth layer as

$$W_{k,2} = U_{k,1} S_{k,1,2} V_{k,1}^\top + U_{k,2} S_{k,2,2} V_{k,2}^\top.$$

$U_{k,1}, V_{k,1}$ represent the two low-rank matrices learned for task 1 and frozen afterwards. $U_{k,2}, V_{k,2}$ represent two low-rank matrices that are added to update the weights, and these will be learned for task 2. $S_{k,1,2}, S_{k,2,2}$ represent the diagonal matrices, which will be learned for task 2. We learn $S_{k,1,2}$, which is a diagonal matrix that assigns weights to factors corresponding to task 1, to include/exclude or favor/suppress frozen factors from previous tasks for the new tasks. We can represent the weights for the kth layer and task t as

$$W_{layer,task} = W_{k,t} = \sum_{i \leq t} U_{k,i} S_{k,i,t} V_{k,i}^\top = \sum_{i<t} \underbrace{U_{k,i}}_{\text{frozen}} S_{k,i,t} \underbrace{V_{k,i}^\top}_{\text{frozen}} + U_{k,t} S_{k,t,t} V_{k,t}^\top, \tag{3}$$

where $U_{k,i}, V_{k,i}$ are frozen for all $i < t$ and $U_{k,t}, V_{k,t}$ and all the $S_{k,i,t}$ are learned for task t. The entire network for task t can be represented as $\mathcal{W}_t = \{U_{k,i}, S_{k,i,t}, V_{k,i}\}_{i \leq t}$. To update the trainable network parameters for task t, we solve the following optimization problem:

$$\min_{U_{k,t}, S_{k,i,t}, V_{k,t}} \sum_{(x,y) \in (\mathcal{X}_t, \mathcal{Y}_t)} \text{loss}(f(x; \mathcal{W}_t[U_{k,t}, S_{k,i,t}, V_{k,t}], y) \;\; k \leq K; i \leq t, \tag{4}$$

where we use $\text{loss}(\cdot, \cdot)$ to denote the loss function and $\mathcal{W}_t[U_{k,t}, S_{k,i,t}, V_{k,t}]$ to indicate the trainable parameters in $\mathcal{W}_t$, while the rest are frozen. We sometimes call $S_{k,i,t}$ a *selector weight matrix/vector* to indicate that its diagonal entries determine the contribution of each factor toward each task/layer weights.

Our proposed continual learning algorithm works as follows. We train the low-rank factors for the given task using the respective training samples. Then we freeze the factors corresponding to the older tasks and only update the new factors and the diagonal matrices. In this manner, the total number of parameters we add in our model is linearly proportional to the rank of the new factors. To keep the complexity of the network small, we seek to achieve good accuracy using small rank for each task update and layer. We summarize our approach in Algorithms 1 and 2.

Note that we do not need to create the weight matrix $W_{k,t}$ for any layer explicitly since we can compute all the steps in forward and backward propagation efficiently using the factorized form of each layer. The size of each layer is determined by the choice of the network architecture. The rank of each layer for every task is a hyper-parameter that we can select according to the tasks at hand. To keep the memory overhead small, we need to use small values for rank increment. Let us denote the rank for $U_{k,t}$ as $r_{k,t}$, which represents the increment rank for kth layer and task t. At the time of test, we can use an appropriate number of factors depending on the task. For instance, if we want to predict output for task 1 then we use first $r_{k,1}$ factors and for task 2 we use $r_{k,1} + r_{k,2}$ factors. We can add new factors in an incremental manner as we add new tasks in the continual learning setup. In the extreme case of rank-1 increments, $r_{k,t} = 1$. In our experiments, we observed that rank-1 updates compete or exceed the performance of existing continual learning methods (see Table 1) and the performance of our method improves further as we increase the rank (see Table 4). Any increase in the rank comes at the expense of an increased memory overhead.

Algorithm 1 Continual learning with low-rank increments (Training)

Input: Data ($\mathcal{X}_1$ and $\mathcal{Y}_1$) for the 1^{st} task.
Set initial rank, r_1.
Initialize weight matrix factors $U_{k,1}, V_{k,1}$ at random and $S_{k,1,1} = 1$.
Learn $U_{k,1}, V_{k,1}$ and $S_{k,1,1}$. ▷ Optimization in equation 4
for $t = 2, 3, ..., T$ **do**
  Input: Training data ($\mathcal{X}_t$ and $\mathcal{Y}_t$) for t^{th} task.
  Initialize low-rank update factors $U_{k,t}, V_{k,t}$.
  Freeze the previous factors $\{U_{k,i}, V_{k,i}\}_{i<t}$.
  Initialize the diagonal entries of $\{S_{k,i,t}\}$ as 1 for $i = t$ and 0 for $i < t$.
  Learn $U_{k,t}, V_{k,t}$ and $S_{k,i,t}$ for all k, $i < t$. ▷ Optimization in equation 4
end for

Algorithm 2 Continual learning with low-rank increments (Testing)

Input: Test data x with task identity t.
Retrieve trained weights: $\mathcal{W}_t = \{U_{k,i}, V_{k,i}, S_{k,i,t}\}$ for all k and $i \leq t$.
Output: Calculate the network output as $f(x, \mathcal{W}_t)$.

4 Experiments and Results

We used different classification tasks on well known continual learning benchmarks to show the significance of our proposed approach.

4.1 Datasets and Task Description

Experiments are conducted on five datasets: Split CIFAR100, Split MNIST, Permuted MNIST, Rotated MNIST, and Split MiniImageNet.
S-MNIST partitions the original MNIST dataset into 5 different 2-way classification tasks, each containing consecutive classes from the original dataset.
P-MNIST creates new tasks by applying a certain random permutation on the pixels of all images in the original dataset. In our experiment, we generate 20 different tasks, each of which corresponds to a certain but different permutation.
R-MNIST is similar to Permuted MNIST, but instead of applying a certain random permutation on the pixels, it applies a certain random rotation to the images in the same tasks. We create 20 different tasks, each corresponds to a certain but different version of rotation from [0, 180] degree interval.
S-CIFAR100 splits the original CIFAR-100 dataset into 20 disjoint sets, each of which, containing 5 classes, is considered as a separate task. The 5 classes in each task is randomly chosen without replacement from the total 100 classes.
S-miniImageNet splits a subset of Imagenet dataset into 20 disjoint sets, each of which, containing 5 classes, is considered as a separate task. The 5 classes in each task is randomly chosen without replacement from the total 100 classes.

4.2 Training Details

Network. We used a three layer fully-connected network with 256-node hidden layers, similar to the network in Chaudhry et al. (2020). We flattened multi-dimensional input image to a 1D vector input. We used ReLU activation for all the layers except the last one. We used sigmoid activation in the last layer for binary classification (S-MNIST) and Softmax for the muticlass classification tasks. We used the same fully connected network for all the tasks with necessary modifications for input and output sizes. Our approach can be used in convolutional networks as well. We report the results using ResNet18 with our approach on S-CIFAR100 dataset in Table 6 (in the appendix).

Factorization. We used the matrix factorization defined in equation 3 in all our experiments. We used the same initial rank or rank increment for all the layers except the last layer.

Rank selection. We empirically selected the rank for the first task,$r_{k,1}$ as 11 based on the experiments on a sample Rotated MNIST task and kept the same value for all the experiments. We then performed rank-1 increment ($r_{k,t}$) for each additional task.

We would like to point that A-GEM and Orthog Subspace use first 3 tasks for hyperparameter tuning. We did not tune our hyperparameters on the test data, rather we choose the parameters which provides better convergence during training. We increment the weight matrices by rank-1 per task; therefore, learning rate and the number of epochs are the only hyperparameters in our experiments.

Optimization. We used orthogonal initialization for the low-rank factors, as described in Saxe et al. (2013). We used all one initialization for the additional factors of the selector matrices $S_{k,t,t}$. We used Adam optimization to update the factors. We used the batch size of 128 for each task.

Performance metric. We use *accuracy* and *forgetting* per task, which are two commonly used metrics in the continual learning literature Chaudhry et al. (2018; 2020), to evaluate the performance of the described methods.

Let $a_{t,j}$ be the test accuracy of task $j < t$ after the model has finished learning task $t \in \{1, ..., T\}$ in a continual manner. The average accuracy A_t after the model has learned task t is defined as

$$A_t = \frac{1}{t} \sum_{j=1}^{t} a_{t,j}. \tag{5}$$

In Figure 2 we show the evolution of average accuracy A_t as t increases. We also show the evolution of task-wise accuracy $a_{t,j}$ in Figure 3, where (t, j) pixel intensity reflects $a_{t,j}$. We report the average accuracy A_T, the average accuracy after the model has learnt every tasks continually, in Table 1.

On the other hand, *forgetting* is the decrease in the accuracy of a task after its training, and after one or several tasks are learned continually. We define the average forgetting F_t as:

$$F_t = \frac{1}{t-1} \sum_{j=1}^{t-1} (a_{l,j} - a_{t,j}) \tag{6}$$

We report the forgetting F_T after the model has learnt all the tasks continually in Table 3. Note that our method performs continual learning without forgetting.

4.3 Comparing Techniques

We compare our method against different state-of-the-art continual learning methods and baselines. **EWC** Kirkpatrick et al. (2017) is a regularization-based method that uses the Fisher Information matrix to estimate posterior of previous tasks to preserve important parameters. **ICARL** Rebuffi et al. (2017) is a memory-based method that uses exemplars and knowledge distillation Hinton et al. (2014) to retain previous knowledge. **AGEM** Chaudhry et al. (2019a) is a memory-based method built upon Lopez-Paz & Ranzato (2017) that uses episodic memory to solve an constrained optimization problem. **Orth. sub.** Chaudhry et al. (2020) learn tasks in different (low-rank) vector subspaces that are kept orthogonal to each other in order to minimize interference.

In addition, we report results for two *non-continual* baseline methods: **Parallel learning** and **Multitask learning**. **Parallel learning** trains independent (smaller) low-rank networks of same size for each task. We report results for three such networks. **Parallel 2** uses rank-2 layers, **Parallel 4** uses rank-4 layers, and **Parallel full** uses a full-rank fully-connected network. Parallel 2 requires approximately the same number of parameters as the rank-1 continual learning network that we use in our experiments; Parallel 4 provides higher network capacity, while requiring fewer parameters than the full-rank network. We can treat the performance of the **Parallel full** approach as the upper limit that we can achieve using the continual method. Finally, **Multitask learning** has been used as a baseline in Chaudhry et al. (2020; 2019a). In multitask learning, we have access to all data to optimize a single network.

4.4 RESULTS

Classification performance and comparison. We report classification results for P-MNIST, R-MNIST, S-MNIST, S-CIFAR100, and S-miniImageNet tasks in Table 1. We also show the results for the comparing techniques. We observe that our method with rank-1 update perform better than all the comparing methods (EWC, ICARL, A-GEM, Orthog Subspace) on R-MNIST, S-CIFAR100 and S-miniImageNet tasks. On S-MNIST and P-MNIST tasks, our method performs close to the best performing Orthog Subspace approach. We also observe that the continual rank-1 update outperforms non-continual Parallel 2 baseline that has similar number of parameters compared to our approach. We perform similar to Parallel 4 baseline that uses nearly twice the number of parameters as our approach. Parallel full acts as an upper limit with the network structure of our choice as it trains independent full rank networks for every task. Multitask learning is another non-continual baseline that uses all the data from all the tasks simultaneously. It appears that the three layer fully-connected network structure does not have enough capacity to learn all the 100 classes at a time for CIFAR100 and miniImageNet classification; therefore, the performance of the multitark learning baseline is quite low for these two datasets. We also tested Resnet18 network, which has significantly more parameters than the network used in Table 1. We report the results for Resnet18 in Table 6 (in the appendix).

ICARL and A-GEM require replay buffer (episodic memory) for each task. Although Orthog Subspace did not use replay buffer for MNIST experiments, it requires replay buffer in their algorithm and used it for S-CIFAR100 and S-miniImageNet experiments. EWC does not require any replay buffer, but it suffers from high forgetting as shown in Figure 3. Our proposed approach does not require any replay buffer and it outperforms other approaches as shown in Table 1.

Table 1: Results on continual learning tasks for S-MNIST, P-MNIST, R-MNIST, S-CIFAR100, and S-miniImageNet. Average test accuracy over five runs (with standard deviation) is shown for all the experiments.

* Orthog subspace does not use replay buffer for MNIST variations.

Method	Replay Buffer	P-MNIST	R-MNIST	S-MNIST	S-CIFAR100	S-miniImageNet
EWC	No	67.9 ($\pm$0.68)	44.5 ($\pm$1.09)	97.3 ($\pm$0.90)	52.7 ($\pm$0.81)	29.3 ($\pm$1.08)
ICARL	Yes	85.4 ($\pm$0.01)	51.2($\pm$ 2.41)	94.1 ($\pm$0.01)	56.9($\pm$0.31)	39.9($\pm$0.27)
A-GEM	Yes	73.9 ($\pm$0.52)	53.4 ($\pm$1.80)	94.4 ($\pm$2.11)	51.3($\pm$1.54)	31.3($\pm$0.89)
Orth sub	Yes*	**86.6** ($\pm$0.79)	80.2 ($\pm$0.41)	**97.8** ($\pm$0.93)	57.8 ($\pm$1.03)	38.1 ($\pm$0.67)
Ours	No	85.6 ($\pm$0.15)	**91.1** ($\pm$0.12)	95.9 ($\pm$0.10)	**65.9** ($\pm$2.16)	**54.7** ($\pm$2.87)
Parallel 2 (r=2)	-	65.3	65.5	97.4	62.8	55.4
Parallel 4 (r=4)	-	86.3	87.4	97.6	65.6	58.6
Parallel full rank	-	95.9	97.3	99.7	73.1	63.1
Multitask learning	-	96.8	97.7	96.7	16.4	4.21

Memory complexity. Since we increment the rank of each layer for each task, we need to compare the total number of parameters in the continual-trained network and the Parallel baselines. Note that if the number of parameters in two approaches is same, we can train one small network per task independently. We report the total number of parameters for our approach and comparing approaches in Table 2. Since we used similar fully connected network structure for all the tasks, we reported results for just Split CIFAR100 classification tasks in Table 2. Although we increase the rank for every task, the increment is so small that even after 20 tasks our total parameter count is nearly $14\times$ smaller the closest comparing approach A-GEM and $140\times$ smaller than the non-continual parallel full rank baseline. We also report the memory overhead in Table 2 for different approaches. To calculate memory overhead, we combine incremental network/task head size and the size of replay buffer (i.e. the episodic memory). Since parallel full-rank network is non-continual, the memory overhead is not reported. We observe that our memory overhead is also significantly smaller than that of comparing approaches.

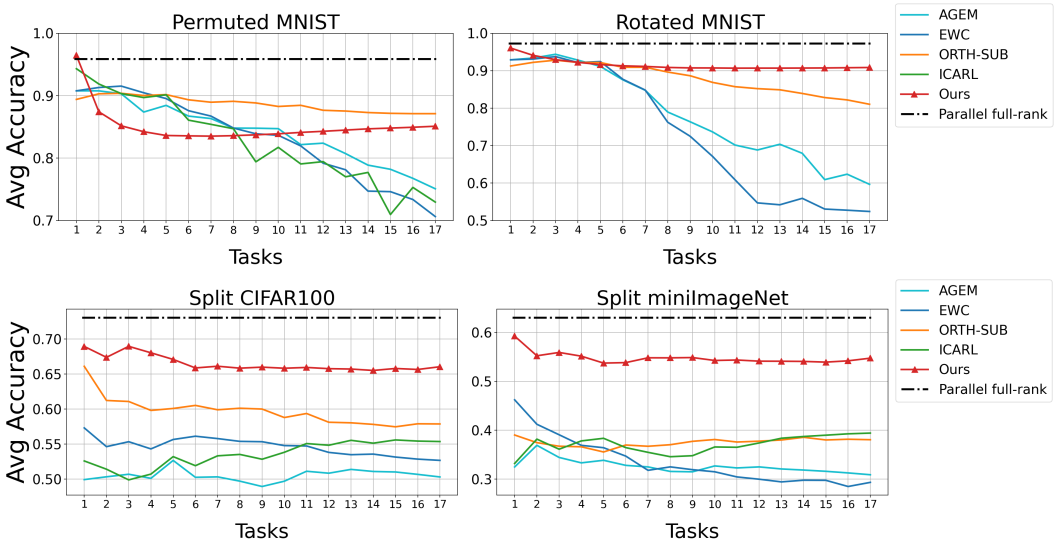

Figure 2: Average test accuracy for different datasets (Permuted MNIST, Rotated MNIST, Split CIFAR100, Split miniImageNet) along different tasks using different algorithms (AGEM,EWC, Orthog. Subspace, ICARL and our approach). Parallel full-rank results corresponds to the case when we train every task on separate full rank networks independently. We showed the average of 20 tasks. It serves as an upper limit for continual learning approaches.

Table 2: Total number of parameters and training memory overhead required by different approaches for continually learning all 20 tasks on Split-CIFAR100 using 3-layer fully-connected network. The numbers inside bracket is relative to the number of parameters of a single full-rank 3-layer fully-connected network. The memory overhead includes both the additional network memory required for each task and the size replay buffers. We use a replay buffer of 20 examples per class per task for A-GEM and Orthog. subspace.

	EWC	Ours	A-GEM	Orthog subspace (w/ replay)	Parallel fullrank
# parameters	0.93M (1.00)	**0.12M (0.14)**	1.76M (1.88)	2.82M (3.03)	19.7M (21.18)
memory overhead	1.71M	**0.170M**	7.90M	9.01M	-

Accuracy vs forgetting. We report the average forgetting of different comparing approaches in Table 3. Our methods and parallel baselines have zero forgetting, whereas all other comparing continual learning methods exhibit some level of forgetting. To better demonstrate the forgetting, in Figure 3, we show the accuracy for the tasks along the entire training procedure. i^{th} row (top-bottom) of the diagram denotes the performance of i tasks on the test sets when we train the i^{th} task. As expected, we can observe that the training performance for the previously learned tasks usually drops with the gradual training of the subsequent tasks specially for the regularization based approach, EWC. However, our algorithm maintains the same performance for the pask tasks as we do not change any previously learned factors. Even orthogonal subspace approach observes such forgetting over some tasks.

Table 3: Average forgetting results for different datasets using different approaches. We report the forgetting in percentage unit (%). We also report the standard deviation.

	EWC	Ours	A-GEM	Orthog subspace (w/ replay)	Parallel fullrank
P-MNIST	25.8 ($\pm$0.70)	**0**	19.6 ($\pm$0.64)	4.49 ($\pm$0.93)	0
R-MNIST	52.9 ($\pm$1.17)	**0**	44.2 ($\pm$1.85)	14.7 ($\pm$0.39)	0
S-CIFAR100	6.96 ($\pm$0.80)	**0**	21.5 ($\pm$2.89)	6.30 ($\pm$0.38)	0
S-miniImageNet	17.3 ($\pm$1.81)	**0**	18.8 ($\pm$1.40)	9.98 ($\pm$0.31)	0

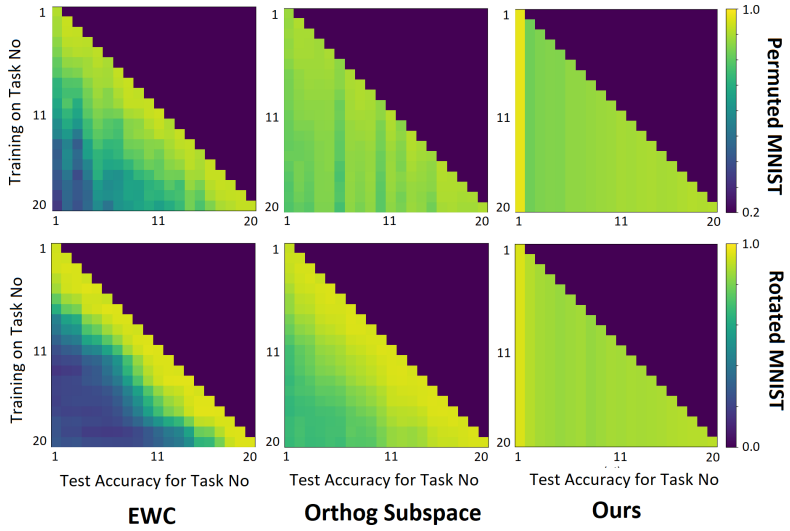

Figure 3: Evolution of task-wise test accuracy on P-MNIST (i,ii,iii) and R-MNIST (iv,v,vi) datasets for EWC (i,iv), Orthogonal Subspace (ii,v), abd Our approach (iii,vi). We can observe from the decrease in the test accuracies that EWC and Orthogonal Subspace forget the previous tasks as they learn new tasks. Our approach does not show any forgetting as we learn new tasks.

Effect of rank. In Table 4, we evaluate the effect of different rank selection for different MNIST datasets using our continual learning approach. We tested the initial rank (rank for the first task) of 1, 6, and 11, keeping the rank increment to 1. We observed that the accuracy increase as the initial rank increases, and we achieve nearly 90% accuracy with initial rank of 11. We also tested different values of rank increment per task and observe that the accuracy increases with larger rank increment. Nevertheless, rank-1 increment provides us comparable or better performance than the comparing techniques as shown in Table 1.

Table 4: Test accuracy for different rank choices of continual learning network and multi-task baseline networks for S-MNIST, P-MNIST, and R-MNIST.

Method	Initial rank,$r_{k,1}$	Rank increment/task, $r_{k,t}$	P-MNIST	R-MNIST	S-MNIST
Continual 1	1	1	74.23	81.57	92.66
Continual 2	6	1	82.21	89.39	94.47
Continual 3	11	1	85.61	91.09	95.92
Continual 4	11	2	90.51	92.76	96.21
Continual 5	11	4	93.84	94.12	97.77

5 Conclusion

We proposed a new continual learning method in which we update the network weights using low rank increments as we learn new tasks. Network layers are represented as a linear combination of low-rank factors. To update the network for a new task, we freeze the factors learned for previous tasks, add a new low-rank (or rank-1) factor, and combine that with the previous factors using a learned combination. The proposed method offered considerable improvement in performance compared to the state-of-the-art methods for continual learning in image classification tasks. In addition, the proposed low-rank incremental continual learning circumvents the use of memory buffer or large memory overhead while achieving zero forgetting.

References

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3366–3375, 2017.

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)*, pp. 139–154, 2018.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in Neural Information Processing Systems*, 32:11816–11825, 2019.

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)*, pp. 532–547, 2018.

Arslan Chaudhry, Ranzato Marc'Aurelio, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations, ICLR*, 2019a.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019b.

Arslan Chaudhry, Naeemullah Khan, Puneet Dokania, and Philip Torr. Continual learning in low-rank orthogonal subspaces. *Advances in Neural Information Processing Systems*, 33, 2020.

Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773. PMLR, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NeurIPS*, 2014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30:6467–6476, 2017.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2019.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32:350–360, 2019.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Daniel L Silver and Robert E Mercer. The task rehearsal method of life-long learning: Overcoming impoverished data. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 90–101. Springer, 2002.

Shixiang Tang, Dapeng Chen, Jinguo Zhu, Shijie Yu, and Wanli Ouyang. Layerwise optimization by gradient decomposition for continual learning. *arXiv preprint arXiv:2105.07561v1*, 2021.

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *International Conference on Learning Representations*, 2020.

Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33, 2020.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

A APPENDIX

Ablation study. In continual learning, we update rank-1 factors, selector matrices, and bias vectors per task. To evaluate the effect of different settings, we trained the networks with (rank-1 factors+bias update, bias update, rank-1 factor update). Test accuracy for different datasets is reported in Table 5. We observe that updating the weight matrices with fixed bias vector for all tasks (rank-1 factors only) or different bias vector for every task (rank-1 factors + bias) provides similar results. For bias update, we learn the rank-11 factors U and V from the first task. Then we only update the bias vector for each task. We observe that updating bias vector alone is sufficient to learn new tasks when the number of samples/task are small (e.g. S CIFAR, S miniImagenet). However, updating bias alone is insufficient when we have a much larger number of samples/task (e.g. P-MNIST, R-MNIST has over 20 times more samples/task).

Table 5: Ablation study to evaluate the effect of updating Test accuracy of of four different datasets under different combinations of selector and bias vector update.

	P-MNIST	R-MNIST	S-CIFAR100	S-miniImageNet
Rank-1 factors + Bias	86.6	91.1	65.9	54.7
Rank-1 factors only	84.8	90.8	64.85	51.97
Bias only	66.1	84.6	63.95	55.61

Convolutional networks. The proposed low-rank increments approach can be generalized to other type of networks and layers as well. For example, convolutional kernels have four-dimensional weight tensors as opposed to the two-dimensional weight matrices of fully connected layers. They are usually formulated as a tensor of output and input channel (C_{out}, C_{in}), and the two dimensions of the convolutional filters (H, W). We reshape the convolutional weight tensors into matrices of size $C_{out} \times C_{in}HW$ and perform similar low-rank updates per task as we described for the fully-connected network in the main paper. We report the results for S-CIFAR-100 dataset with Resnet18 architecture. For each convolutional layers, we reshaped and decomposed the convolution weight tensors into the same low-rank factors described in (3) and performed low-rank updates per tasks. We report the results in Table 6. Instead of using a fixed value for rank at each layer as we did in the fully-connected network setup, we used rank size that is proportional to the size of $C_{out,i}$ at i^{th} convolutional layer. We select initial rank = $0.1\, C_{out,i}$ for the first task and incremental rank = $0.02\, C_{out,i}$ for the subsequent continual learning tasks.

Table 6: Test accuracy and forgetting results on split CIFAR-100 dataset using ResNet18 architecture with different continual and non-continual approaches. With convolutional ResNet18 structure every approach perform better, but our method outperforms the comparing approaches.
** EWC and ICARL results in this table are reported from Chaudhry et al. (2020)

	S-CIFAR-100	
	Accuracy	Forgetting
EWC**	43.2 (±2.77)	26 (±2)
ICARL**	46.4 (±1.21)	16 (±1)
AGEM	60.34 (± 2.05)	11.0 (± 2.88)
Ortho sub	63.42 (± 1.82)	8.37 (± 0.71)
Ours	**68.46** (±2.52)	**0**
Parallel full-rank	73.1	0
Multitask learning	70.2	0