Incremental Task Learning with Incremental Rank Updates

Rakib Hyder[1], Ken Shao[1], Boyu Hou[1], Panos Markopoulos[2], Ashley Prater-Bennette[3], and M. Salman Asif[1]

[1] University of California Riverside
[2] Rochester Institute of Technology
[3] Air Force Research Laboratory

Abstract. Incremental Task learning (ITL) is a category of continual learning that 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, ITL methods use episodic memory, parameter regularization, masking and pruning, or extensible network structures. In this paper, we propose a new incremental task learning framework based on low-rank factorization. In particular, we represent the network weights for each layer as a linear combination of several rank-1 matrices. To update the network for a new task, we learn a rank-1 (or low-rank) 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- and mask-based approaches. Our code will be available at `https://github.com/CSIPlab/task-increment-rank-update.git`

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 [25]. 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 [10,25]. In such a training process, a network often forgets the previously learned tasks; this effect is termed *catastrophic forgetting* [21,26].

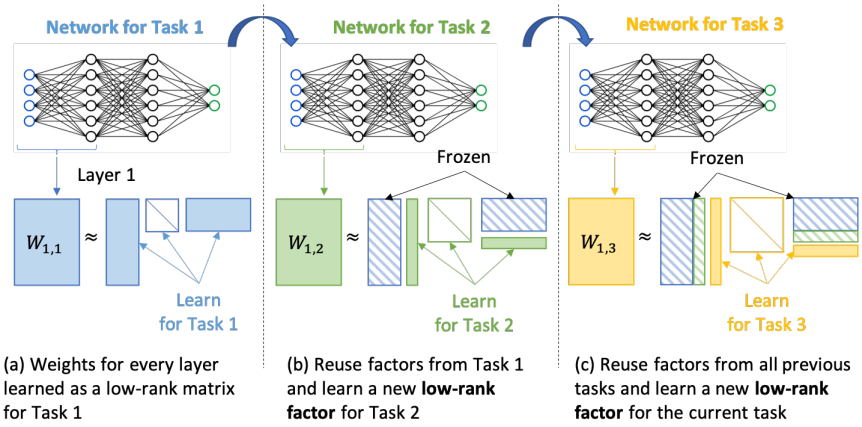

Fig. 1: An overview of our proposed method for continual learning via low-rank network updates. We first represent (and learn) the weight matrix (or tensor) for each layer as a product of low-rank matrices. To train a network for new tasks without forgetting the earlier tasks, we reuse the factors from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

Incremental task learning is a subcategory of 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 [23,16,3,2,4,6,28,29,12]. 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 [8]. It has been a popular continual learning setup even in the very recent literature [7,14,11,36,31,40]. ITL finds its application in setups where task id is available during inference; for instance, tasks performed under different weather/light/background conditions and we know the changes, or tasks learned on different data/classes where we know the task id.

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 from the previous task ($\mathcal{W}_{t-1}$) to $\mathcal{W}_t$ such that

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

ITL 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. Dynamic architecture approaches have the potential to achieve *zero forgetting*, using $\mathcal{W}_t$ for testing data for task t; 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 ITL that updates network weights using rank-1 (or low-rank) increments for every new task. 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. As we are learning separate diagonal matrices for every task, we can achieve zero forgetting during inference. 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 method shares same inherent limitation of ITL (i.e. the requirement of task-id during inference). In addition, 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 significantly lower than a single network. Furthermore, as we learn separate diagonal matrices for each task, we can maintain high performance even if the network reaches full rank with a large number of tasks.

2 Background and Related Work

Incremental task learning (ITL) [10,34] 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 [21]" is observed during this learning process. Catastrophic forgetting occurs when 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. Some of these approaches are especially designed for ITL whereas others are designed for more general continual learning setup.

Regularization-based approaches [15,23,16] 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) [15] estimates the importance of parameters using Fisher Information matrix; Variational Continual Learning (VCL) [23] approximates the posterior distribution of the parameters using variational inference; Learning without Forgetting (LwF) [16] regularizes the current loss with soft targets taken from previous tasks using knowledge distillation [13]. GCL [5] mixes rehearsal with knowledge distillation and regularization to mitigate catastrophic forgetting. A number of recently proposed methods force weight updates to belong to the null space of the feature covariance [37,35].

Memory-based approaches [27,28,8,9,35] usually use memory and replay/rehearsal 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 [27] 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) [28] 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 [8] projects the gradient on the current minibatch by using an external episodic memory of patterns from previous experiences as an optimization constraint; ER-Ring [9] jointly trains new task data with that of the previous tasks.

Dynamic network architectures [30,19,39,38,33,7,41] 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) [30] leverage prior knowledge via lateral connections to previously learned features; PackNet [19] iteratively assigns parameter subsets to consecutive tasks by constituting binary masks. SupSup [39] also finds masks in order to assign different subsets of the weights for different tasks. BatchEnsemble [38] learns on separate rank-1 scaling matrices for each task which are then used to scale weights of the shared network. HAT [33] incorporates task-specific embeddings for attention masking. [24] also proposes task-conditioned hypernetworks for continual learning.

[20] proposes nonoverlapping sets of units being active for each task. Piggyback [18] learns binary masks on an existing network to provide good performance on new tasks. [1] proposes task specific convolutional filter selection for continual learning. The mask-based methods listed above provide excellent results for continual learning, but they require a significantly large number of parameters to represent the masks for each task. A factorization-based approach was proposed in [22] that performs automatic rank selection per task for variational inference using Indian Buffet process. The method requires significantly large rank increments per task to achieve high accuracy; in contrast, our method uses a learning-based approach to find rank-1 increments and reuse old factors with the learned selector weights. ORTHOG-SUBSPACE [7] learns 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 Incremental Task Learning via Rank Increment

We focus on the incremental task learning setup in which we seek to train a network for T tasks. The main difference between incremental task 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 incremental task 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 seek to develop an ITL 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

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

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 $\mathcal{W}_t = \{U_{k,i}, S_{k,i,t}, V_{k,i}\}_{i \le t}$. To update the trainable network parameters for task t, we solve the following optimization problem:

$$\begin{aligned} \min_{U_{k,t}, S_{k,i,t}, V_{k,t}} \sum_{(x,y) \in (\mathcal{X}_t, \mathcal{Y}_t)} & \text{loss}(f(x; \mathcal{W}_t[U_{k,t}, S_{k,i,t}, V_{k,t}]), y) \\ & \text{for all } k \le K \text{ and } i \le t, \end{aligned} \tag{4}$$

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 ITL 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 network complexity 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 ITL 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 ITL methods (see Table 1) and the performance of our method improves further as we increase the rank (see Table 5). Any increase in the rank comes at the expense of an increased memory overhead.

Algorithm 1 ITL with rank-1 increments (Training)

Input: Data ($\mathcal{X}_1$ and $\mathcal{Y}_1$) for the 1^{st} task.
Set initial rank, r_1.
Initialize weight factors $U_{k,1}, V_{k,1}$ at random and $S_{k,1,1}$ as an identity marix.
Learn $U_{k,1}, V_{k,1}$ and $S_{k,1,1}$. ▷ Optimization in (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 $i < t$. ▷ Optimization in (4)
end for

Algorithm 2 ITL with rank-1 increments (Inference)

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 four datasets: Split CIFAR100, Permuted MNIST, Rotated MNIST, and Split MiniImageNet.
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. In the first set of experiments, we used a three layer (fully-connected) multilayer perceptron (MLP) with 256-node hidden layers, similar to the network in [7]. 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 Softmax for the muticlass classification tasks. We used the same 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 and S-miniImageNet dataset in Table 6.
Factorization and rank selection. We used the matrix factorization defined in (3) in all our experiments. 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 AGEM 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 [32]. 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 metrics. We use *accuracy* and *forgetting* per task, which are two commonly used metrics in the continual learning literature [6,7], 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 incremental manner. The average accuracy A_t after the model has learned task t is defined as $A_t = \frac{1}{t}\sum_{j=1}^{t} a_{t,j}$. 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 incrementally. We define the average forgetting F_t as $F_t = \frac{1}{t-1}\sum_{j=1}^{t-1}(a_{j,j} - a_{t,j})$.

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 incrementally, in Table 1. We report the forgetting F_T after the model has learnt all the tasks incrementally in Table 2. Note that our method performs incremental task learning without forgetting.

4.3 Comparing Techniques

We compare our method against different state-of-the-art ITL methods. **EWC** [15] is a regularization-based method that uses the Fisher Information matrix to estimate posterior of previous tasks to preserve important parameters. **ICARL** [27] is a memory-based method that uses exemplars and knowledge distillation [13] to retain previous knowledge. **AGEM** [8] is a memory-based

method built upon [17] that uses episodic memory to solve an constrained optimization problem. **ER-Ring** [9] is another memory-based method that jointly trains on new task data with that of the previous tasks. **Orth. sub.** [7] learn tasks in different (low-rank) vector subspaces that are kept orthogonal to each other in order to minimize interference. Other than the above mentioned approaches, we compared with masked based approaches which, like our approach, also fall under dynamic architecture category. **HAT** [33] that incorporates task-specific embeddings for attention masking. **PackNet** [19] that iteratively assigns subsets of a single binary mask to each task. The mask-based approaches utilize the redundancy of the network parameters to represent different tasks with different masked versions of the same network weights. We also present comparisons with some recent methods: **IBP-WF** [22], **DER** [5] and **Adam-NSCL** [37], in terms of average accuracy for one experiment on two datasets.

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 MLP. Parallel 2 requires approximately the same number of parameters as the rank-1 ITL 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 ITL methods. Finally, **Multitask learning** has been used as a baseline in [7,8]. In multitask learning, we have access to all data to optimize a single network.

4.4 Results with Three-Layer MLP

Classification performance and comparison. We report classification results for P-MNIST, R-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, AGEM, HAT, PackNet, Orthog Subspace) on R-MNIST, S-CIFAR100 and S-miniImageNet tasks using significantly fewer number of parameters. Our method performs close to Orthog Subspace on P-MNIST tasks.

We also observe that the proposed 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. Table 1 suggests that our ITL method can learn complex tasks such as CIFAR100 and miniImageNet classification with a three layer MLP, whereas multitask learning (which is solving 100-class classification problem) fails with such a simple network. We also tested Resnet18 network, which has significantly more parameters than the network used in Table 1. The results for Resnet18 are presented in Table 6.

Table 1: Average test accuracy of ITL for P-MNIST, R-MNIST, S-CIFAR100, and S-miniImageNet with three layer MLP. Standard deviation of test accuracy over five runs is shown in parenthesis for some of the experiments.

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

Method	Replay Buffer	P-MNIST	R-MNIST	S-CIFAR100	S-miniImageNet
EWC [15]	No	67.9 (±0.68)	44.5 (±1.09)	52.7 (±0.81)	29.3 (±1.08)
ICARL [27]	Yes	85.4 (±0.01)	51.2(± 2.41)	56.9(±0.31)	39.9(±0.27)
AGEM [8]	Yes	73.9 (±0.52)	53.4 (±1.80)	51.3(±1.54)	31.3(±0.89)
HAT [33]	No	**90.1**(±1.60)	89.1(±2.51)	64.8 (±0.32)	47.0 (±0.88)
PackNet [19]	No	90.0(±0.24)	88.4(±0.37)	63.7(±0.41)	45.1(±1.05)
Orth sub [7]	Yes*	86.6 (±0.79)	80.2 (±0.41)	57.8 (±1.03)	38.1 (±0.67)
DER [5]	Yes	–	–	48.21	33.19
Adam-NSCL[37]	No	–	–	64.26	47.32
IBP-WF [22]	No	–	–	53.22	40.52
Ours	No	85.6 (±0.15)	**91.1** (±0.12)	**65.9** (±2.16)	**54.7** (±2.87)
Parallel 2 (r=2)	-	65.3	65.5	62.8	55.4
Parallel 4 (r=4)	-	86.3	87.4	65.6	58.6
Parallel fullrank	-	95.9	97.3	73.1	63.1
Multitask	-	96.8	97.7	16.4	4.21

We present the task-wise test performance for some of the comparing approaches on P-MNIST, R-MNIST, S-CIFAR100 and S-miniImageNet datasets in Figure 2. We observe that as we train new tasks, task-wise performance drops for the comparing approaches, especially for P-MNIST and R-MNIST.

ICARL and AGEM 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 a replay buffer, and it outperforms other approaches in Table 1.

Accuracy vs forgetting. We report the average forgetting of different comparing approaches in Table 2. Our method, mask-based approaches (HAT and PackNet) and parallel baselines have zero forgetting, whereas all other comparing 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 past tasks as we do not change any previously learned factors. Even orthogonal subspace approach observes such forgetting over some tasks.

Table 2: Average forgetting results corresponding to Table 1 for different datasets using different approaches. We report the forgetting in percentage unit (%). We also report the standard deviation over 5 experiments for some methods.

	EWC	AGEM	Orthog subspace	DER	Adam-NSCL	Parallel fullrank, HAT, PackNet **Ours**, IBP-WF
P-MNIST	25.8 (±0.70)	19.6 (±0.64)	4.49 (±0.93)	-	-	0
R-MNIST	52.9 (±1.17)	44.2 (±1.85)	14.7 (±0.39)	-	-	0
S-CIFAR100	6.96 (±0.80)	21.5 (±2.89)	6.30 (±0.38)	10.6	8.5	0
S-miniImageNet	17.3 (±1.81)	18.8 (±1.40)	9.98 (±0.31)	20.11	11.23	0

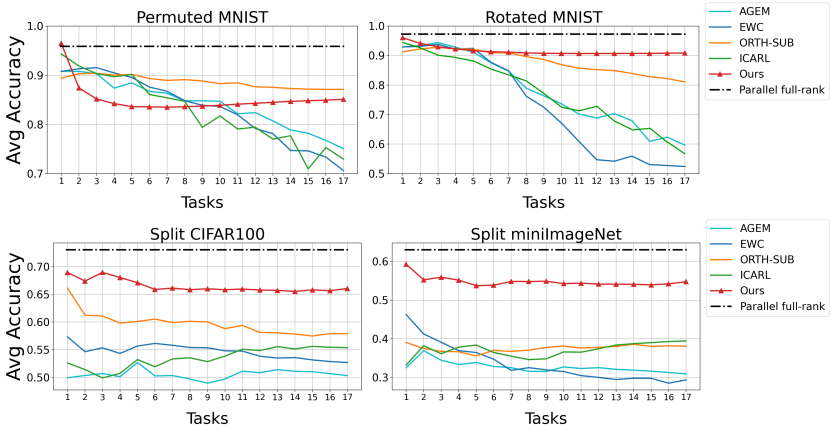

Fig. 2: Average test accuracy for different datasets (Permuted MNIST, Rotated MNIST, Split CIFAR100, Split miniImageNet) along different tasks using different algorithms (AGEM,EWC, Orthog. Subspace, ICARL and our approach). We use three layer MLP here. Parallel full-rank results corresponds to the case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

Memory complexity. Our method increments the rank of each layer for each task; therefore, we compare the total number of parameters in the incrementally 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 total number of parameters and replay buffer size for different methods in Table 3. Since we used similar fully connected network structure for all the tasks, we report results for Split CIFAR100 experiments. Although we increase the rank for every task, the increment is small enough that even after 20 tasks our total parameter count remains smaller than all other methods.

We also report the number of parameters used by mask-based zero forgetting algorithms (HAT and PackNet) to learn 20 different tasks on different datasets in Table 4. We can observe that our approach outperforms HAT and PackNet for R-MNIST, S-CIAR100 and S-miniImageNet with a significantly smaller number of

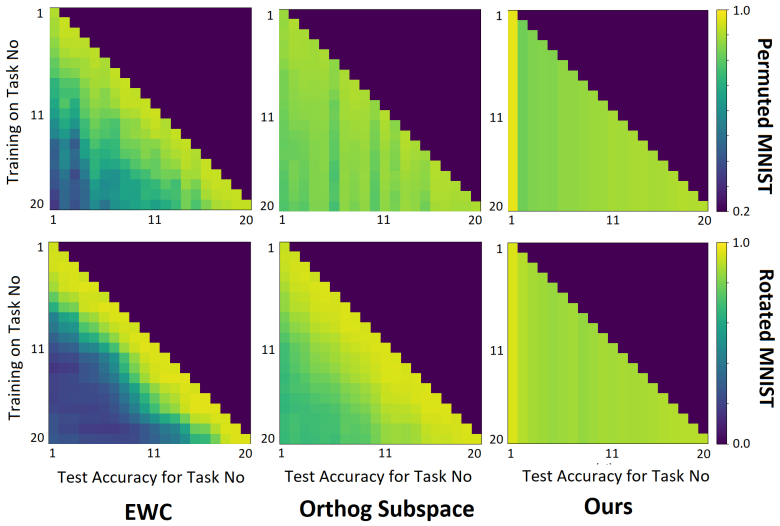

Fig. 3: Evolution of task-wise test accuracy on P-MNIST (first row) and R-MNIST (second row) datasets for EWC, Orthogonal Subspace, and Our approach. We can observe from the decrease in the test accuracy that EWC and Orthogonal Subspace forget the previous tasks as they learn new tasks. Our approach does not show any forgetting as we learn new tasks.

Table 3: Number of parameters and buffer size in ITL methods with 3-layer MLP.

	Ours	IBP-WF	EWC	AGEM	Ortho Sub	DER	Adam-NSCL	Para. full.
# params.	**0.17M**	0.23M	0.93M	1.76M	2.82M	0.88M	0.88M	19.7M
buffer size	0	0	1.71M	7.90M	9.01M	6.14M	0	0

parameters. Even though all the approaches use the same network, our approach uses rank-1 factors that require a significantly smaller number of parameters for incremental learning of tasks. Note that P-MNIST and R-MNIST experiments require the same number of parameters.

Effect of rank. In Table 5, we evaluate the effect of different rank selection for different MNIST datasets using our ITL 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: Number of parameters used by different zero-forgetting algorithms (HAT, PackNet, and Ours) using 3-layer MLP.

Method	P/R-MNIST	S-CIFAR100	S-miniImageNet
HAT	0.33M	0.89M	5.51M
PackNet	0.26M	0.83M	5.50M
Ours	0.11M	0.17M	0.72M

Table 5: Test accuracy for different rank choices of the proposed ITL approach and multi-task baseline networks for P-MNIST and R-MNIST. Initial rank is $r_{k,1}$ and rank increment/task is $r_{k,t}$.

Setup	1	2	3	4	5
$(r_{k,1}, r_{k,t})$	(1,1)	(6,1)	(11,1)	(11,2)	(11,4)
P-MNIST	74.23	82.21	85.61	90.51	93.84
R-MNIST	81.57	89.39	91.09	92.76	94.12
# parameters	0.09M	0.1M	0.11M	0.14M	0.2M

4.5 Results with ResNet18

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 MLP in the main paper. We report the results for S-CIFAR-100 and S-miniImageNet datasets 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. For most of the comparing techniques, results from [7] are reported since we use the same architecture and dataset. For missing comparisons, we trained the models using same procedure as outlined in [7].

Instead of using a fixed value for rank at each layer as we did in the MLP setup, we used rank size that is proportional to the size of $C_{out,i}$ at i^{th} convolutional layer because the weights for different layers of ResNet18 are different in size. We select initial rank $= 0.1\, C_{out,i}$ for the first task and incremental rank $= 0.02\, C_{out,i}$ for the subsequent incremental tasks.

The results in Table 6 show that the performance of every method improves with the convolutional ResNet18 structure over the 3-layer MLP. Nevertheless, our method outperforms the comparing approaches for both datasets. Adam-NSCL [37] gets better results on CIFAR100, but it requires 11.21M parameters (compared to 1.33M parameters required by our method).

Table 6: Comparison of test accuracy and forgetting for split CIFAR-100 and split miniImageNet datasets using ResNet18 architecture.

Method	S-CIFAR-100		S-miniImageNet	
	Accuracy	Forgetting	Accuracy	Forgetting
EWC [15]	43.2 (±2.77)	26 (±2)	34.8 (±2.34)	24 (±4)
ICARL [27]	46.4 (±1.21)	16 (±1)	44.2	24.64
AGEM [8]	60.34 (±2.05)	11.0 (±2.88)	42.3 (±1.42)	17 (±1)
ER-Ring [9]	59.6 (±11.9)	14 (±1)	49.8 (±2.92)	12 (±1)
Ortho sub [7]	63.42 (±1.82)	8.37 (±0.71)	51.4 (±1.44)	10 (±1)
DER [5]	67.16	8.95	57.81	14.70
Adam-NSCL [37]	**74.31**	9.47	57.92	13.42
IBP-WF [22]	68.25	0	55.84	0
Ours	68.46 (±2.52)	**0**	**59.26** (±1.15)	**0**
Parallel full-rank	92.7	0	94.5	0
Multitask learning	70.2	0	65.1	0

Effect of updating last few layers. We performed an experiment on S-CIFAR-100 where we factorize last L layers of the ResNet18 architecture keeping the rest of the network fixed at trained weights on Task 1. Updating last $L = \{1, 2, 3, 4, 5\}$ layers provide average accuracy of $\{34.38, 34.99, 53.41, 57.08, 65.03\}$, respectively. This result suggests that updating last few layers may suffice since the initial layers merely work as a feature extractor.

5 Conclusion

We proposed a new incremental task 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 ITL in image classification tasks. In addition, the proposed low-rank ITL circumvents the use of memory buffer or large memory overhead while achieving zero forgetting.

The need for task ID knowledge is a general limitation of our and other ITL methods. Such methods can be useful for incremental multitask learning where task ID is available during inference but training data is only available in a short window. Extending this method to class incremental learning (which does not require task ID) is an important problem for future work.

Acknowledgments. This material is based upon work supported in part by by Air Force Office of Scientific Research (AFOSR) awards FA9550-21-1-0330, FA9550-20-1-0039, Office of Naval Research (ONR) award N00014-19-1-2264, and National Science Foundation (NSF) award CCF-2046293. Approved for Public Release by AFRL; Distribution Unlimited: Case Number AFRL-2021-4063

Corresponding author: M. Salman Asif (sasif@ucr.edu)

References

1. Abati, D., Tomczak, J., Blankevoort, T., Calderara, S., Cucchiara, R., Bejnordi, B.E.: Conditional channel gated networks for task-aware continual learning. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition. pp. 3930–3939 (2020)
2. Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 139–154 (2018)
3. Aljundi, R., Chakravarty, P., Tuytelaars, T.: 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)
4. Aljundi, R., Lin, M., Goujaud, B., Bengio, Y.: Gradient based sample selection for online continual learning. Advances in Neural Information Processing Systems **32**, 11816–11825 (2019)
5. Buzzega, P., Boschini, M., Porrello, A., Abati, D., Calderara, S.: Dark experience for general continual learning: a strong, simple baseline. Advances in neural information processing systems **33**, 15920–15930 (2020)
6. Chaudhry, A., Dokania, P.K., Ajanthan, T., Torr, P.H.: Riemannian walk for incremental learning: Understanding forgetting and intransigence. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 532–547 (2018)
7. Chaudhry, A., Khan, N., Dokania, P., Torr, P.: Continual learning in low-rank orthogonal subspaces. Advances in Neural Information Processing Systems **33** (2020)
8. Chaudhry, A., Marc'Aurelio, R., Rohrbach, M., Elhoseiny, M.: Efficient lifelong learning with a-gem. In: International Conference on Learning Representations, ICLR (2019)
9. Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P.K., Torr, P.H., Ranzato, M.: On tiny episodic memories in continual learning. arXiv preprint arXiv:1902.10486 (2019)
10. Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., Tuytelaars, T.: A continual learning survey: Defying forgetting in classification tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence (2021)
11. Deng, D., Chen, G., Hao, J., Wang, Q., Heng, P.A.: Flattening sharpness for dynamic gradient projection memory benefits continual learning. Advances in Neural Information Processing Systems **34** (2021)
12. Farajtabar, M., Azizan, N., Mott, A., Li, A.: Orthogonal gradient descent for continual learning. In: International Conference on Artificial Intelligence and Statistics. pp. 3762–3773. PMLR (2020)
13. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. NeurIPS (2014)
14. Hurtado, J., Raymond-Saez, A., Soto, A.: Optimizing reusable knowledge for continual learning via metalearning. Advances in Neural Information Processing Systems **34** (2021)
15. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al.: Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences **114**(13), 3521–3526 (2017)
16. Li, Z., Hoiem, D.: Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence **40**(12), 2935–2947 (2017)

17. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continual learning. Advances in neural information processing systems **30**, 6467–6476 (2017)
18. Mallya, A., Davis, D., Lazebnik, S.: Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 72–88 (2018)
19. Mallya, A., Lazebnik, S.: 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)
20. Masse, N.Y., Grant, G.D., Freedman, D.J.: Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. Proceedings of the National Academy of Sciences **115**(44), E10467–E10475 (2018)
21. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation, vol. 24, pp. 109–165. Elsevier (1989)
22. Mehta, N., Liang, K., Verma, V.K., Carin, L.: Continual learning using a bayesian nonparametric dictionary of weight factors. In: International Conference on Artificial Intelligence and Statistics. pp. 100–108. PMLR (2021)
23. Nguyen, C.V., Li, Y., Bui, T.D., Turner, R.E.: Variational continual learning. In: International Conference on Learning Representations (2018)
24. von Oswald, J., Henning, C., Sacramento, J., Grewe, B.F.: Continual learning with hypernetworks. In: International Conference on Learning Representations (2019)
25. Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S.: Continual lifelong learning with neural networks: A review. Neural Networks **113**, 54–71 (2019)
26. Ratcliff, R.: Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychological review **97**(2), 285 (1990)
27. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 2001–2010 (2017)
28. Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., Tesauro, G.: Learning to learn without forgetting by maximizing transfer and minimizing interference. In: International Conference on Learning Representations (2019)
29. Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., Wayne, G.: Experience replay for continual learning. Advances in Neural Information Processing Systems **32**, 350–360 (2019)
30. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016)
31. Saha, G., Garg, I., Roy, K.: Gradient projection memory for continual learning. In: International Conference on Learning Representations (2021)
32. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120 (2013)
33. Serra, J., Suris, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: International Conference on Machine Learning. pp. 4548–4557. PMLR (2018)
34. Silver, D.L., Mercer, R.E.: 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)
35. Tang, S., Chen, D., Zhu, J., Yu, S., Ouyang, W.: Layerwise optimization by gradient decomposition for continual learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9634–9643 (2021)

36. Veniat, T., Denoyer, L., Ranzato, M.: Efficient continual learning with modular networks and task-driven priors. In: International Conference on Learning Representations (2021)
37. Wang, S., Li, X., Sun, J., Xu, Z.: Training networks in null space of feature covariance for continual learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 184–193 (2021)
38. Wen, Y., Tran, D., Ba, J.: Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In: International Conference on Learning Representations (2020)
39. Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., Farhadi, A.: Supermasks in superposition. Advances in Neural Information Processing Systems **33** (2020)
40. Yin, H., Li, P., et al.: Mitigating forgetting in online continual learning with neuron calibration. Advances in Neural Information Processing Systems **34** (2021)
41. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: International Conference on Learning Representations (2018)