Lifelong Learning via Online Leverage Score Sampling

Anonymous Authors[1]

Abstract

In order to mimic the human ability of continual acquisition and transfer of knowledge across various tasks, a learning system needs the capability for life-long learning, effectively utilizing the previously acquired skills. As such, the key challenge is to transfer and generalize the knowledge learned from one task to other tasks, avoiding interference from previous knowledge and improving the overall performance. In this paper, within the continual learning paradigm, we introduce a method that *effectively forgets* the less useful data samples continuously across different tasks. The method uses statistical leverage score information to measure the importance of the data samples in every task and adopts frequent directions approach to enable a life-long learning property. This effectively maintains a constant training size across all tasks. We first provide some mathematical intuition for the method and then demonstrate its effectiveness with experiments on variants of MNIST and CIFAR100 datasets.

1. Introduction

It is a typical practice to design and optimize machine learning (ML) models to solve a single task. On the other hand, humans, instead of learning over isolated complex tasks, are capable of generalizing and transferring knowledge and skills learned from one task to another. This ability to remember, learn and transfer information across tasks is referred to as lifelong learning or continual learning (Thrun & Mitchell, 1995; Hassabis et al., 2017; Parisi et al., 2019). The major challenge for creating ML models with lifelong learning ability is that they are prone to catastrophic forgetting (McClelland et al., 1995; McCloskey & Cohen, 1989). ML models tend to forget the knowledge learned from previous tasks when re-trained on new observations corresponding to a different (but related) task. Specifically when a deep neural network (DNN) is fed with a sequence of tasks, the ability to solve the first task will decline significantly after training on the following tasks. The typical structure of DNNs by design does not possess the capability of preserving previously learned knowledge without interference between tasks or catastrophic forgetting. There have been different approaches proposed to address this issue and they can be broadly categorized in three types:

I) **Regularization**: It constrains or regularizes the model parameters by adding some terms in the loss function that prevent the model from deviating significantly from the parameters important to earlier tasks. Typical algorithms include elastic weight consolidation (EWC) (Kirkpatrick et al., 2017) and continual learning through synaptic intelligence (SynInt) (Zenke et al., 2017).

II) **Architectural modification**: It revises the model structure successively after each task in order to provide more memory and additional free parameters in the model for new task input. Recent examples in this direction are progressive neural networks (Rusu et al., 2016) and dynamically expanding networks (Yoon et al., 2018).

III) **Memory replay**: It stores data samples from previous tasks in a separate memory buffer and retrains the new model based on both the new task input and the memory buffer. Popular algorithms here are gradient episodic memory (GEM) (Lopez-Paz & Ranzato, 2017), incremental classifier and representation learning (iCaRL) (Rebuffi et al., 2017).

Among these approaches, regularization is particularly prone to saturation of learning when the number of tasks is large. The additional / regularization term in the loss function will soon lose its competency when important parameters from different tasks are overlapped too many times. Modifications on network architectures like progressive networks resolve the saturation issue, but do not scale as number and complexity of tasks increase. The scalability problem is also present when using memory replay and often suffer from high computational and memory costs.

In this paper, we propose a novel approach to lifelong learning with DNNs that addresses both the learning saturation and high computational complexity issues. In this method, we progressively compresses the input information learned thus far along with the input from current task and form more efficiently condensed data samples. The compression technique is based on the statistical leverage scores measure, and it uses frequent directions idea in order to connect the series of compression steps for a sequence of

tasks. Our approach resembles the use of memory replay since it preserves the original input data samples from earlier tasks for further training. However, our method does not require extra memory for training and is cost efficient compared to most memory replay methods. Furthermore, unlike the importance assigned to model specific parameters when using regularization methods like EWC or SynInt, we assign importance to the training data that is relevant in effectively learning new tasks, while forgetting less important information.

2. Online Leverage Score Sampling (OLSS)

Before presenting the idea, let's first setup the problem:

Let $\{(A_1, B_1), (A_2, B_2), ..., (A_i, B_i), ...\}$ represent a sequence of tasks, each task consists of n_i data samples and each sample has a feature dimension d and an output dimension m, i.e., input $A_i \in \mathbb{R}^{n_i \times d}$ and true output $B_i \in \mathbb{R}^{n_i \times m}$. Here, we assume the feature and output dimensions are fixed for all tasks [1]. The goal is to train a DNN over the sequence of tasks and ensure it performs well on all of them. Here, we consider that the network's architecture stays the same and the tasks are received in a sequential manner. Formally, with f representing a DNN, our objective is to minimize the loss [2]:

$$\min_f \left\| f(A) - B \right\|_2^2 \text{ where } A = \begin{bmatrix} A_1 \\ A_2 \\ ... \\ A_i \\ ... \end{bmatrix} \text{ and } B = \begin{bmatrix} B_1 \\ B_2 \\ ... \\ B_i \\ ... \end{bmatrix}. \tag{1}$$

Under this setup, let's look at some existing models:
Online EWC trains f on task (A_i, B_i) with a loss function containing additional penalty terms $\min_f \left\| f(A_i) - B_i \right\|_2^2 + \sum_{j=1}^{i-1} \Lambda_j$ and each Λ_j is defined as the change of important parameters (using Fisher information matrix) in f with respect to the jth task.

GEM keeps an extra memory buffer containing data samples from each of the previous tasks $\mathcal{M}_k$ with $k < i$, it trains on the current task (A_i, B_i) with a regular loss function $\min \left\| f(A_i) - B_i \right\|_2^2$, but subject to inequalities on each update of f, $\langle \frac{\partial \left\| f_\theta(A_i) - B_i \right\|_2^2}{\partial \theta}, \frac{\partial \left\| f_\theta(A_{\mathcal{M}_k}) - B_{\mathcal{M}_k} \right\|_2^2}{\partial \theta} \rangle \geq 0$ for all $k < i$.

[1] If we know apriori that the feature or output dimensions are different, we could choose a presumed larger value of d and m. In lifelong learning our aim is to solve successive problems with some degree of overlap. As such, the feature and output dimensions being the same across tasks is not overly strict.

[2] Here, we represent a generic Euclidean loss term. However, this could take the form of any typical formulation in terms of l_1-loss, l_2-loss or cross-entropy loss as commonly used in classification problems.

The new approach OLSS is to find an approximation of A in a streaming manner, i.e., to form an $\hat{A}_i$ to approximate $[A_1 \; A_2 \; \cdots \; A_i]^T$ such that the resulting

$$\hat{f}_i := \arg\min_f \left\| f(\hat{A}_i) - \hat{B}_i \right\|_2^2$$

is likely to perform on all tasks as good as

$$f_i^* := \arg\min_f \left\| f([A_1 \; A_2 \; \cdots \; A_i]^T) - [B_1 \; B_2 \; \cdots \; B_i]^T \right\|_2^2. \tag{2}$$

To avoid extra memory and computation cost during the training process, we restrict the approximate $\hat{A}_i$ to have the same number of rows as the current task A_i.

Equation (1) and (2) represent nonlinear least squares problems. It is to be noted that a nonlinear least squares problem can be solved with an approximation deduced from an iteration of linear least squares problems with $J^T J \Delta\theta = J^T \Delta B$ where J is the Jacobian of f at each update (Gauss-Newton Method). Besides this technique, there are various approaches in addressing this problem. Here we adopt a cost effective simple randomization technique - leverage score sampling, which has been used extensively in solving large scale linear least squares and low rank approximation problems (Woodruff, 2014; Cohen et al., 2017).

Statistical Leverage Score

Definition 1 *(Drineas et al., 2012) Given a matrix $A \in \mathbb{R}^{n \times d}$ with $n > d$, let U denote the $n \times d$ matrix consisting of the d left singular vectors of A, and let $U_{(i,:)}$ denote the i-th row of U, then the statistical leverage score of the i-th row of A is defined as $\left\| U_{(i,:)} \right\|_2^2$ for $i \in \{1, ..., n\}$.*

Statistical leverage scores define the relevant non-uniformity structure of a matrix and a higher score indicates a heavier weight of the row contributing to the non-uniformity of the matrix; it has been widely used for constructing a randomized sketch of a matrix (Drineas et al., 2012; Woodruff, 2014). In our case, given an input matrix A, we will compute the leverage score of each row, then sample the rows with probability proportional to the scores.

Using leverage score sampling, we are able to select the important samples given a dataset. The remaining problem is to embed it in a sequence of tasks. In order to achieve this, we make use of the concept of frequent directions.

Frequent Directions

Frequent directions extends the idea of frequent items in item frequency approximation problem to a matrix (Liberty, 2013; Ghashami et al., 2016; Teng & Chu, 2018). Given a matrix $A \in \mathbb{R}^{n \times d}$ whose rows are received one by one

and a space parameter ℓ, the algorithm considers the first 2ℓ rows in A and shrinks its top ℓ orthogonal vectors by the same amount to obtain an $\ell \times d$ matrix; then combines them with the next ℓ rows in A for the next iteration, repeat the procedure until reaching the final sketch of dimension $\ell \times d$. Frequent directions algorithm is targeted at finding a low rank approximation on a continuously expanding matrix. This is well suited for a continuous stream of data (tasks) within the lifelong learning setting. We present the step by step procedure of performing leverage score sampling together with compression using frequent directions idea in Algorithm 1. In our setting, we append the new task data samples to the existing buffer set and perform leverage score sampling to form a new buffer set and then train on it, this process is repeated for the entire sequence of tasks.

Main Algorithm

Algorithm 1 OLSS

Input: A sequence of tasks $\{(A_1, B_1), ..., (A_i, B_i), ...\}$ with $A_i \in \mathbb{R}^{n_i \times d}$ and $B_i \in \mathbb{R}^{n_i \times m}$; initialization of the model parameters; a space parameter ℓ i.e., number of samples to pass in the model for training. It can be set as n_i or even smaller after receiving the i-th task, which avoids extra memory and computations during training.
Output: A trained neural network on a sequence of tasks.
Step 1 Initialize a buffer set $S = \{\hat{A}, \hat{B}\}$ where both $\hat{A}$ and $\hat{B}$ are empty.
Step 2 While the ith task is presented:
Step 3 If $\hat{A}$ and $\hat{B}$ are empty:
Step 4 set $\hat{A} = A_i$ and $\hat{B} = B_i$,
Step 5 else:
Step 6 set $\hat{A} = \begin{bmatrix} \hat{A} \\ A_i \end{bmatrix}$ and $\hat{B} = \begin{bmatrix} \hat{B} \\ B_i \end{bmatrix}$.
Step 7 Perform SVD: $[U, \Sigma, V^T] = svd(\hat{A})$.
Step 8 Randomly select ℓ rows of $\hat{A}$ and $\hat{B}$ without replacement based on probability $\|U_{j,:}\|_2^2 / \|U\|_F^2$ for $j \in \{1, ..., n_i + \ell\}$ (or $j \in \{1, ..., n_i\}$ when $i = 1$) and set them as $\hat{A}$ and $\hat{B}$ respectively.
Step 9 Train the model with $\hat{A} \in \mathbb{R}^{\ell \times d}$ and $\hat{B} \in \mathbb{R}^{\ell \times m}$.
Step 10 End

When n_i is large, the SVD (singular value decomposition) of matrix $\hat{A} \in \mathbb{R}^{(n_i+\ell) \times d}$ in **Step 6** is computationally expensive, we could use a streaming SVD method to speed up the process if ℓ is chosen much smaller than n_i. In that case the computational cost for SVD could be reduced from $O((n_i + \ell)d^2)$ to $O(\log_2(n_i + \ell)\ell d^2)$ (assuming $d < \ell < n_i$). In addition, there exists various efficient ways to approximate the leverage scores (Drineas et al., 2012; Rudi et al., 2018) which would further reduce the computational cost.

Remark: A major concern with this algorithm is that leverage scores is a linear measure, i.e., the selected samples capture the important information embedded linearly in the data matrix which may not fully represent the importance of the data samples. Another related issue is that the nonlinear information probably depend on the structure of f, the DNN. As such, there may be some underlying dependency of a data sample's importance on the DNN architecture. We leave this open as a future research direction.

3. Experiments

We evaluate the performance of the proposed algorithm OLSS on three classification tasks used as benchmarks in related prior work.

- **Rotated MNIST** (Lopez-Paz & Ranzato, 2017): a variant of the MNIST dataset of handwriten digits (LeCun et al., 1998), the digits in each task are rotated by a fixed angle between $0°$ to $180°$. The experiment is on 20 tasks and each task consists of $60,000$ training and $10,000$ testing samples.
- **Permutated MNIST** (Kirkpatrick et al., 2017): a variant of the MNIST dataset (LeCun et al., 1998), the digits in each task are transformed by a fixed permutation of pixels. The experiment is on 20 tasks and each task consists of $60,000$ training and $10,000$ testing samples.
- **Incremental CIFAR100** (Rebuffi et al., 2017; Zenke et al., 2017): a variant of the CIFAR object recognition dataset with 100 classes (Krizhevsky, 2009). The experiment is on 20 tasks and each task consists of 5 classes; each task consists of $2,500$ training and 500 testing samples. Where, each task introduces a new set of classes; for a total number of 20 tasks, each new task concerns examples from a disjoint subset of 5 classes.

In the original setting of (Lopez-Paz & Ranzato, 2017), a softmax layer is added to the output vector which only allows entries representing the 5 classes in the current task to output values larger than 0. In our setting, we allow the entries representing all the past occurring classes to output values larger than 0. We believe this is a more natural setup for lifelong learning.

The DNN used for rotated and permuted MNIST is an MLP with 2 hidden layers and each with 400 units; whereas a ResNet18 is used for the incremental CIFAR100 experiment. We train 5 epochs with batch size 200 on rotated and permuted MNIST datasets and 10 epochs with batch size 100 on incremental CIFAR100. In all experiments we compare the following algorithms: I) A simple SGD predictor, II)

EWC (Kirkpatrick et al., 2017), III) GEM (Lopez-Paz & Ranzato, 2017) and IV) OLSS (ours).

In all the algorithms, we use a plain SGD optimizer. All algorithms were implemented based on the publicly available code from the original authors of the GEM paper (Lopez-Paz & Ranzato, 2017). The regularization and memory hyper-parameters in EWC and GEM were set as described in (Lopez-Paz & Ranzato, 2017). The space parameter for our OLSS algorithm was set to be equal to the number of samples in each task; the learning rate for each algorithm was determined through a grid search on $\{0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$.

Results

Comparing across all the algorithms, we summarize the average test accuracy on the learned tasks in Figure 1 (see appendix Figure 2 for the change in the test accuracy at the first task, as more tasks are learned.) and the computational costs for each algorithm in Table 1. As observed from the figures, across the three benchmarks, OLSS and GEM achieve similar accuracy and significantly outperform both EWC and simple SGD training. Nevertheless, GEM demands much higher computational resources (see Table 1) as the algorithm requires a constraint validation step and a potential gradient projection step to correct for constraint violations across all previously learned tasks during training (see Section 3 in (Lopez-Paz & Ranzato, 2017)). In detail, for GEM, the time complexity is proportional to the product of the number of samples kept in the memory buffer, the number of parameters in the model and the number of iterations required to converge. In contrast, OLSS requires a SVD (or QR factorization) to compute the leverage scores for each task which can be achieved in a time complexity proportional to the product of the square of the number of features and the number of data samples, and is much less compared to GEM. As observed in appendix Figure 2, OLSS shows robustness to catastrophic forgetting of the first task with positive backward transfer across all three datasets while learning the remaining sequence of tasks. In the case of rotated and permuted MNIST, OLSS is the most robust method.

Table 1. Wall Clock Time (s)

	ROTATED MNIST	PERMUTED MNIST	INCREMENTAL CIFAR100
SGD	158	152	780
EWC	944	896	1213
GEM	8688	8846	17868
OLSS	496	455	1363

As presented in appendix Figure 3, after training on the whole sequence of tasks, both GEM and OLSS are able to preserve the accuracy for most tasks on rotated and permuted MNIST. In contrast, it is hard to preserve the accuracy of the previously trained tasks on CIFAR100 for all algorithms. As we noted earlier, EWC exhibits a saturation issue when the number of tasks increases. This may hold for most regularization methods in order to achieve continual learning, as they target constraining the model parameters successively, thereby limiting the model capacity.

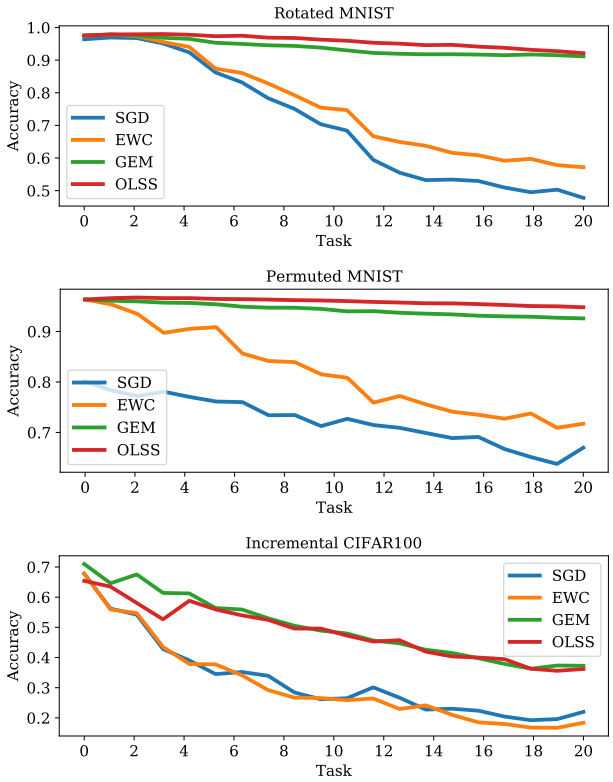

Figure 1. Evolution of average test accuracy across all the learned tasks after training on a sequence of tasks. (E.g., the accuracy value at Task $= 10$ means the average test accuracy on Task $1 - 10$ after training the model for 10 consecutive tasks.)

4. Conclusions

We presented a new approach in addressing the lifelong learning problem with deep neural networks. It is inspired by the randomization and compression techniques typically used in statistical analysis. We combined a simple importance sampling technique - leverage score sampling with the frequent directions concept and developed an online effective forgetting or compression mechanism that enables lifelong learning across a sequence of tasks. Despite its simple structure, the results on MNIST and CIFAR100 experiments show its effectiveness as compared to recent state of the art.

References

Cohen, M. B., Musco, C., and Musco, C. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1758 – 1777, 2017.

Drineas, P., Magdon-Ismail, M., Mahoney, M. W., and Woodruff, D. P. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13:3441–3472, 2012.

Ghashami, M., Liberty, E., Phillips, J. M., and Woodruff, D. P. Frequent directions: Simple and deterministic matrix sketching. *SIAM Journal of Computing*, 45:1762 – 1792, 2016.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron Review*, 95(2):245 – 258, 2017.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., T, Ramalho, Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *PNAS*, 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009.

LeCun, Y., Cortes, C., and Burges, C. J. The mnist database of handwritten digits. *URL: http://yann.lecun.com/exdb/mnist/*, 1998.

Liberty, E. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

Lopez-Paz, D. and Ranzato, M. A. Gradient episodic memory for continual learning. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102:419 – 457, 1995.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104 – 169, 1989.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2017.

Rudi, A., Calandriello, D., Carratino, L., and Rosasco, L. On fast leverage score sampling and optimal learning. In *Advances in Neural Information Processing Systems*, pp. 5677 – 5687, 2018.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv:1606.04671*, 2016.

Teng, D. and Chu, D. Fast frequent directions for low rank approximation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Thrun, S. and Mitchell, T. Lifelong robot learning. *Robotics and Autonomous Systems*, 15:25 – 46, 1995.

Woodruff, D. Sketching as a tool for numerical linear algebra. *Foundations and Trends ® in Theoretical Computer Science*, 10:1–157, 2014.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

A. Further experimental results

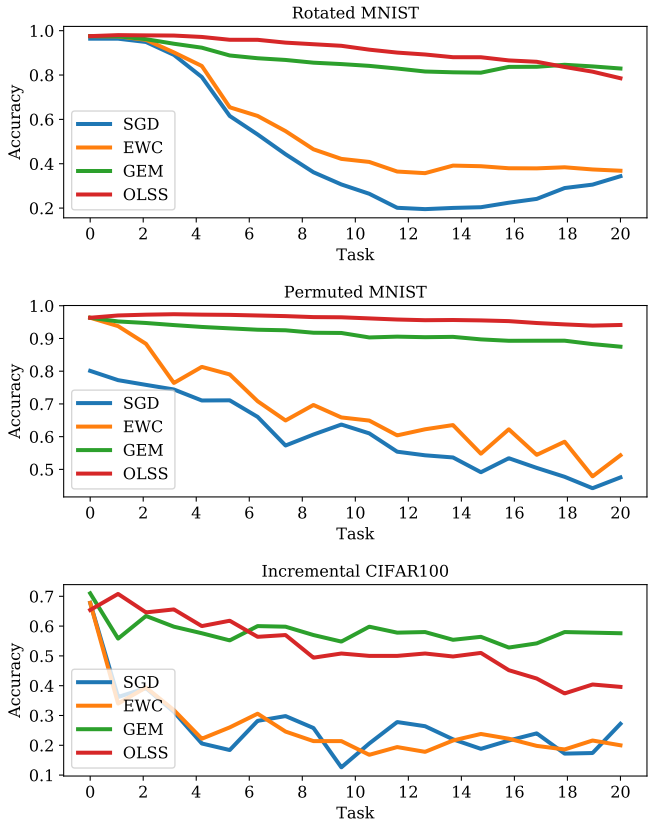

Figure 2. Evolution of test accuracy for the first task after training on a sequence of tasks. (E.g., the accuracy value at Task = 10 means the accuracy of Task 1 after training the model for 10 consecutive tasks.)

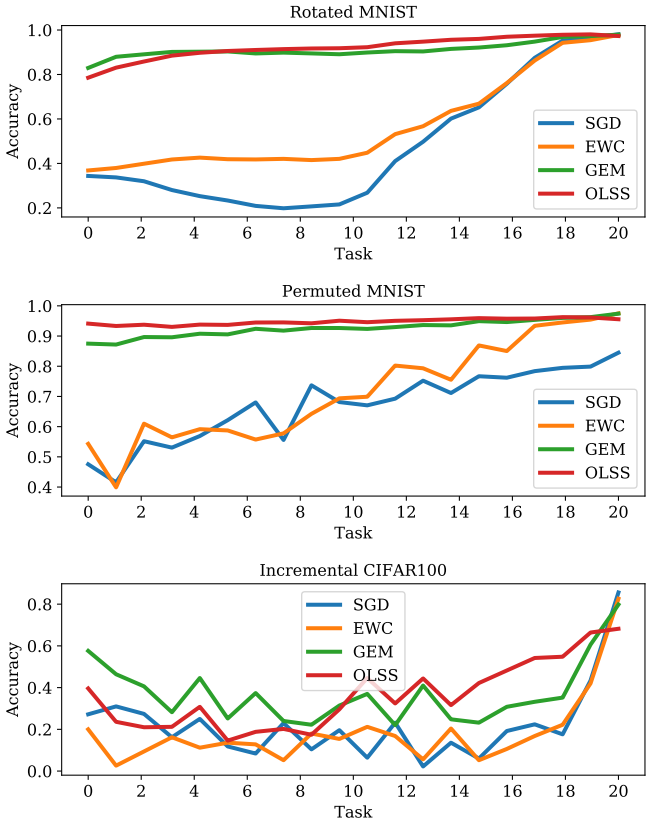

Figure 3. Accuracy of each task after training sequentially on all tasks. (E.g., the accuracy value at Task = 10 means the accuracy of Task 10 after training the model on all tasks sequentially.)