Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira
Georgia Institute of Technology
{yenchang.hsu,ycliu,anita.ramasamy,zkira}@gatech.edu

Abstract

Continual learning has received a great deal of attention recently with several approaches being proposed. However, evaluations involve a diverse set of scenarios making meaningful comparison difficult. This work provides a systematic categorization of the scenarios and evaluates them within a consistent framework including strong baselines and state-of-the-art methods. The results provide an understanding of the relative difficulty of the scenarios and that simple baselines (Adagrad, L2 regularization, and naive rehearsal strategies) can surprisingly achieve similar performance to current mainstream methods. We conclude with several suggestions for creating harder evaluation scenarios and future research directions. The code is available at https://github.com/GT-RIPL/Continual-Learning-Benchmark.

1 Introduction

While current learning-based methods can achieve high performance on tasks, they only perform well when the testing data is similarly distributed as the training data. In other words, they cannot adapt continuously in dynamic environments where situations can significantly change. Such adaptation is desirable for any intelligent system, and is the hallmark of learning in biological systems. One approach to this problem is continual learning, where models are updated incrementally as data streams in. However, deep neural networks, which are currently state of art for many applications, are known to suffer catastrophic interference or forgetting when incrementally updated through gradient-based methods. This leads to the model forgetting how to solve old tasks after being exposed to a new one due to interference caused by parameter updates.

To address this problem, several approaches have been proposed and a number of experimental methodologies (i.e. datasets, learning curricula, and architectures) have been used for evaluation. In this paper, we argue that the current set of evaluations have significant limitations, including lack of uniformity across the different experimental methodologies, lack of hyper-parameter tuning of reasonable baselines under similar tuning budgets as the proposed methods, and simplicity of the tasks (e.g. short task queues). Towards this end, we make several contributions in this paper: 1) A categorization of a large number of experimental methodologies into a few canonical settings along with a comparison of their difficulty, 2) A uniform but flexible framework for generating scenarios under this categorization and systematic evaluation of current state of art methods, and 3) Demonstration that very simple baselines can be surprisingly effective if used properly and result in comparable or better performance against the current state of art. We have released our framework (written in PyTorch) to enable fair and uniform evaluation to aid the community, and conclude with some suggested modifications to the scenarios to increase realism of the evaluation.

Table 1: The continual learning scenarios categorized by the difference between the old and new task. $P(X)$: The distribution of input data. $P(Y)$: The distribution of target labels. $\{Y_1\} \neq \{Y_2\}$: The labels are from a disjoint space which is differentiated by task identity. S: Single-headed model. M: Multi-headed model. I: Known task identity.

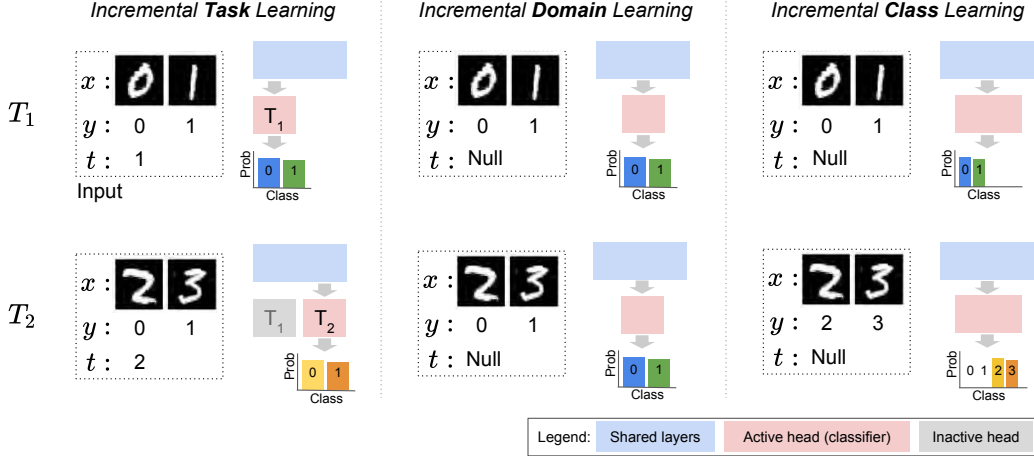| Learning scenario | Old Task (T_1) versus New Task (T_2) | | | Remark |
	$P(X_1) \neq P(X_2)$	$P(Y_1) \neq P(Y_2)$	$\{Y_1\} \neq \{Y_2\}$	
Non-incremental learning				
Incremental domain learning	✓			S
Incremental class learning	✓	✓		S
Incremental task learning	✓	✓	✓	M, I

Figure 1: The three continual learning scenarios generated by Split MNIST. In each sub-figure, the left dotted rectangle represents the inputs for training, in that (x, y, t) means (input image, target class label, input task identity). The right side illustrates the neural network model and the predicted $P(Y)$ of the model. The color of each bar in the categorical distribution maps to a specific output node in the classifier. Note that Split MNIST generates five splits in sequence (0/1, 2/3, 4/5, 6/7, 8/9) for task T_1 to T_5 while here we only demonstrate the differences between T_1 and T_2.

2 Generating Task Sequences for Evaluating Continual Learning

2.1 Existing Experimental Methodologies

In order to evaluate continual learning methods, scenarios are commonly generated from datasets using two operations: permutation and splitting. The typical source dataset is MNIST [1], an image dataset of hand-written digits. The Permuted MNIST experiment [2] involves ten-digit classification, where each task consists of different permutations of the pixels in the images. The number of different permutations represents the length of the task sequence. This evaluation scenario is widely adopted [3, 4, 5, 6, 7, 8, 9], despite criticism that it is less challenging in terms of forgetting [10]. Another typical scenario, the Split MNIST experiment, was initially introduced in a multi-headed form where the ten digits are split into five two-class classification tasks (the model has five output heads, one for each task) [9, 8, 6], and the task identity (1 to 5) is given at testing time. This scenario is argued to be easier since the selection of output head is given by the task identity [10]. Farquhar and Gal [10] propose a single-headed variant which does not require task identity, where it always requires the model to make a prediction over all classes (digits 0 to 9). Such single-headed Split MNIST is known as incremental class learning [11, 12, 13]. Van and Tolias [14] propose another variant of single-headed Split MNIST, where the model always predicts over two classes instead of ten classes. Furthermore, the similar multi-headed/single-headed strategies in Split MNIST can apply to Permuted MNIST [14] resulting in many combinations. These scenarios are used in different works, and therefore there is a lack of coherent comparison. This paper addresses this problem by providing a systematic interpretation of the differences between an old task and a new one (see Section 2.2).

2.2 A Categorization of Current Scenarios

We now provide a uniform categorization for these different combinations. The challenge of continual learning comes from the differences between tasks. The differences can be described by a shifting input/output distribution, and whether the input/output share the same representation space. For notation, we use $T_1 = (X_1, Y_1, t_1)$ to represent the old task, where $X = \{x_i\}_{1 \leq i \leq n}$ is a set of learning samples, $Y = \{y_i\}_{1 \leq i \leq n}$ is a set of output targets, and $t \in \mathbb{Z}$ is a task identity shared among all samples in a task. We use $T_2 = (X_2, Y_2, t_2)$ to denote a new task. A task sequence, $\{T_1, T_2, ..., T_k\}$, represents a scenario for continual learning. The types of differences between tasks are summarized in Table 1. We categorize differences into the following (note that [14] independently developed a similar categorization which we align with but more formally describe):

Incremental Domain Learning: First, we discuss change in the marginal probability distribution of inputs $P(X)$, specifically $P(X_1) \neq P(X_2)$. The input distribution (domain) difference has been discussed extensively in the setting of transfer learning [15], primarily as a domain adaptation problem [16]. Unlike domain adaptation, which aims to transfer knowledge from an old task to a new task where only the performance of the new task is considered, the continual learning setting aims to keep performance on old tasks while achieving reasonable performance on the new one as well using a single model. This domain difference can be caused by the permutation and splitting strategies mentioned previously. In fact, the most widely adopted Permuted MNIST experiment [3, 4, 5, 6, 7, 8, 9] generates such a domain difference. However, the inputs generated by the random permutation protocol is highly uncorrelated [9, 10]; thus they can not represent all possible scenarios. To generate a scenario with better-correlated tasks, one should avoid random permutation to keep the original spatial correlation between image pixels, allowing the possibility of shared features among tasks. This requirement can be fulfilled by a variant of the Split MNIST experiment, where ten digits are split into five binary classification tasks and the model has only a binary classifier (single-headed). Such a scenario is illustrated in the middle column of Figure 1. The requirement of using a single-headed model is essential to control for other types of differences. Specifically, the single-headed model ensures the same output space $\{Y_1\} = \{Y_2\}$ (task identity becomes unnecessary), and the equal amount of MNIST digits ensures the output distributions of the binary classification are the same ($P(Y_1) = P(Y_2)$). As a result, the only difference in the middle column of Figure 1 is that the input images for label $\{0, 1\}$ shift from digit 0/1 to digit 2/3.

Incremental Class Learning: The second scenario relates to multiclass classification. Each task in the sequence contains an exclusive subset of classes in a dataset. $P(X_1) \neq P(X_2)$ is true by the nature of this setting. All class labels are in the same naming space (single-headed) and the number of output nodes equals the number of total classes in the task sequence. Due to the multiclass property, $P(Y_1) \neq P(Y_2)$ is a natural consequence. The right-most column in Figure 1 demonstrates how split-dataset strategy [11, 10] generates this scenario. The permutation strategy can also generate the task sequence [14]. In the latter case, each permuted digit represents a new class, so the total number of classes is multiplied by the number of permutations, e.g., total 10×10 classes in a ten Permuted MNIST experiment.

Incremental Task Learning: In the last scenario, the output spaces are disjoint between tasks, denoted as $\{Y_1\} \neq \{Y_2\}$. This definition makes $P(Y_1) \neq P(Y_2)$ true as well, while $P(X_1) \neq P(X_2)$ is generally true since the semantic classes differ. The differences between the output spaces are the output dimension and their associated semantic meaning. For example, the old task can be a classification problem of 5 classes while the new task is a regression task of a single value. To allow a model to produce an output for a specific task, a model requires task-specific output components which are selected by additional information, the task identifier t. A typical neural network for this scenario has a multi-headed output layer (one head for each task) [9]. At testing time, only the head matching the t will be activated to make predictions. One common approach to generate this scenario is having multiple datasets (ex: MNIST, CIFAR10, SVHN, AudioSet, CUB-200, etc.) and use one of them in one task [17, 18, 19, 20]. The splitting and permutation strategies can also generate task sequences for this scenario, illustrated in Figure 1 and Appendix Figure 2. The prior works mentioned in this paragraph commonly have the task identity given during testing; thus the experiments in this work follow the same setting.

Table 2: The average accuracy (%, higher is better) of all seen tasks after learning the task sequence generated by Split MNIST (Figure 1). The *Memory* column means whether a method uses a memory mechanism, which further divides the methods into two groups in the comparison. The total static memory overhead is controlled to be the same among L2, Naive rehearsal, Naive rehearsal-C, online EWC, SI, MAS, GEM, and DGR. Each value is the average of 10 runs.

	Method	Memory	Incremental task learning	Incremental domain learning	Incremental class learning
Baselines	Adam		93.46 ± 2.01	55.16 ± 1.38	19.71 ± 0.08
	SGD		97.98 ± 0.09	63.20 ± 0.35	19.46 ± 0.04
	Adagrad		98.06 ± 0.53	58.08 ± 1.06	19.82 ± 0.09
	L2		98.18 ± 0.96	66.00 ± 3.73	22.52 ± 1.08
	Naive rehearsal	✓	99.40 ± 0.08	95.16 ± 0.49	90.78 ± 0.85
	Naive rehearsal-C	✓	$\mathbf{99.57} \pm 0.07$	$\mathbf{97.11} \pm 0.34$	$\mathbf{95.59} \pm 0.49$
Continual learning methods	EWC		97.70 ± 0.81	58.85 ± 2.59	19.80 ± 0.05
	Online EWC		98.04 ± 1.10	57.33 ± 1.44	19.77 ± 0.04
	SI		98.56 ± 0.49	64.76 ± 3.09	19.67 ± 0.09
	MAS		99.22 ± 0.21	68.57 ± 6.85	19.52 ± 0.29
	LwF		99.60 ± 0.03	71.02 ± 1.26	24.17 ± 0.33
	GEM	✓	98.42 ± 0.10	96.16 ± 0.35	92.20 ± 0.12
	DGR	✓	99.47 ± 0.03	95.74 ± 0.23	91.24 ± 0.33
	RtF	✓	$\mathbf{99.66} \pm 0.03$	$\mathbf{97.31} \pm 0.11$	$\mathbf{92.56} \pm 0.21$
Offline (upper bound)			99.52 ± 0.16	98.59 ± 0.15	97.53 ± 0.30

3 Experiments

We now describe the experimental configuration. Here, we use the MNIST dataset with the splitting strategy (Figure 1) to generate the three continual learning scenarios in Table 1. The standard train/test split was used, with 60k training images ($\sim$6k per digit) and 10k test images ($\sim$1k per digit). The preprocessing of images includes zero padding to 32x32 pixels and a standard normalization to zero mean with unit variance. No other data augmentation, (e.g. random translation) is applied.

For a fair comparison, all methods use the same neural network architecture, which is a multi-layered perceptron with two hidden layers of 400 nodes each, followed by a softmax output layer. Both hidden layers use ReLU for the activation function. The loss function is a standard cross-entropy for classification in all methods and scenarios. All models are trained for 4 epochs per task with mini-batch size 128 using the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate$= 0.001$) as the default unless explicitly described. In all experiments, the optimizer is never reset.

We use three baseline strategies. The most common baseline used in prior work is a neural network sequentially trained on all tasks in the standard way, in that the **parameters learned from old tasks are fine-tuned to the new task**. Such a model is usually optimized with Adam [21], but here we use different optimizers including Adam, SGD, and Adagrad [22]. In fact, we show that Adam is in general a poor choice for this task. The latter two optimizers use 0.01 for the learning rate without momentum in all scenarios. Another baseline, **L2 regularization**, prevents the parameters from deviating too much from those previously learned. Note that this is similar to EWC in that the identity matrix replaces the Fisher information matrix. In other work this is only evaluated in one specific scenario (permuted MNIST) with limited length (3) of task sequence [3]. Similar to other regularization methods, the L2 regularization requires tuning of the single-valued regularization coefficient, which is done by a grid search [3, 9].

The third baseline is a **naive rehearsal strategy**, which is sometimes called experience replay. The model has a small replay buffer to store a fraction of previous data randomly. While training a new task, each mini-batch is constructed by an equal amount (64/64) of new data and the rehearsal data. The buffer size is predefined and fixed to match the space overhead introduced by online EWC and SI (#parameter $\approx (1024 \times 400 + 400 \times 400) \times 2 = 1,139,200$, ignoring small overhead), which converts to 1.1k images when the pixel value is saved in a 32-bit floating number (named Naive

rehearsal). Additionally, with the same memory space, it can store more images when compression is used. One naive compression is using an 8-bit integer to represent the pixel value; thus 4.4k images can be stored (named Naive rehearsal-C). For the buffer management, all tasks seen so far have an equal amount of images in the buffer while keeping the total number the same. This management is similar to iCaRL [11], except that we randomly pick the images for staying in the buffer.

For comparison, we pick several popular methods (EWC[3], online EWC[23], SI[9], MAS[24], LwF[25], GEM[5]) and state-of-the-art rehearsal-based methods (DGR[8], RtF[14]) with generative model. The hyperparameter is tuned by a grid search, and the results with the best setting are reported. The total static memory overhead is controlled to be the same among Naive rehearsal, Naive rehearsal-C, L2, online EWC, SI, MAS, GEM, and DGR. RtF has only half the overhead of DGR since its classification model is shared with its generative model. The hyperparameter of EWC, online EWC, SI, and MAS is tuned with grid search. We use the results from Ven and Tolias [14], which provides an analysis and comparison developed concurrently with our work, for LwF, DGR, and RtF since the same model and training procedures are used.

4   Results and Discussion

Three interesting points can be seen in the results in Table 2. First, Adagrad and L2 achieve better performance than online EWC and similar to SI. This shows that while Adam is popularly used for this task, Adagrad is more appropriate. This is possibly due to the fact that it results in small updates for parameters frequently used for past tasks. Second, naive rehearsal achieves performance similar to state-of-the-art methods with the same space overhead, and performs much better than online EWC and SI, especially in the incremental class scenario. This highlights the limitation of regularization-based methods and raises a question about the benefit of using a generative model, which is more difficult to train. The third is the obvious trend of difficulty among the three scenarios. The easiest one is incremental task learning, while the incremental class learning is harder than the incremental domain learning.

A similar trend also happens in the scenarios generated by the permutation strategy, which can be seen in Appendix Table 3. In that case, SI and online EWC are significantly better than Adagrad in only one of the three scenarios (incremental class), although all three methods present poor performance in that scenario. One aspect that is not apparent from these results, but is illustrated in Appendix Figure 4, is that EWC and SI variants require significant hyper-parameter tuning, with a wide gap between their worst and best performance (no such tuning was done for Adagrad). This would be difficult to tune automatically in real-world scenarios, however. One interesting cross-table comparison is that the performance of our six baselines in the scenarios generated by permutation (Appendix Table 3) is generally comparable or higher than the same scenarios generated by splitting (Table 2), although the Permuted MNIST scenarios have a larger number of classes and tasks. Such a result indicates that the permutation strategy creates simpler scenarios.

The last highlight is the comparison between Adagrad and EWC. We note that the difference between their performance is not significant in most of the experimental settings, yet EWC requires knowing the boundaries of a task to calculate Fisher information and store the parameters before switching to the next task. This creates a requirement that makes EWC less applicable than Adagrad in a real-world scenario, where the task boundaries are usually not available. Other regularization-based methods, such as SI and MAS, also suffer from the same limitation.

The strong baseline performance, especially in the incremental task learning, does not mean a scenario is solved. One can easily increase the difficulty by using a harder dataset or using a set of datasets to extend the length of a task sequence, as demonstrated in Appendix Section B. Under longer task sequences, regularization-based methods continue to degrade over time leading to questions about whether they fundamentally address catastrophic forgetting. Indeed, biological systems continue to learn new tasks with very little degradation in performance, even when that task has not been seen for a while. One avenue of research in this respect is to take a closer look at continual learning for tasks where feature sharing is possible. How such shared features can be learned when possible, or augmented with new features when tasks distributions differ significantly, is an open question. We also argue that more future effort should be put into scenarios that do not require knowing the task identity (incremental domain/class). Such scenarios are not only harder but also closer to a real scenario where the prior information about task selection is usually weak.

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[3] 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*, 2017.

[4] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, 2017.

[5] David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[6] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *International Conference for Learning Representations*, 2018.

[7] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810*, 2018.

[8] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, 2017.

[9] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017.

[10] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[11] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[12] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[13] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *arXiv preprint arXiv:1806.08568*, 2018.

[14] Gido M van de Ven and Andreas S Tolias. Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*, 2018.

[15] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge & Data Engineering*, 2009.

[16] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[17] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. *ECCV*, 2018.

[18] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[19] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *AAAI Conference on Artificial Intelligence*, 2018.

[20] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *ICLR*, 2018.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2015.

[22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[23] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4528–4537, 2018.

[24] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. *ECCV*, 2018.

[25] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, pages 614–629. Springer, 2016.

Appendices

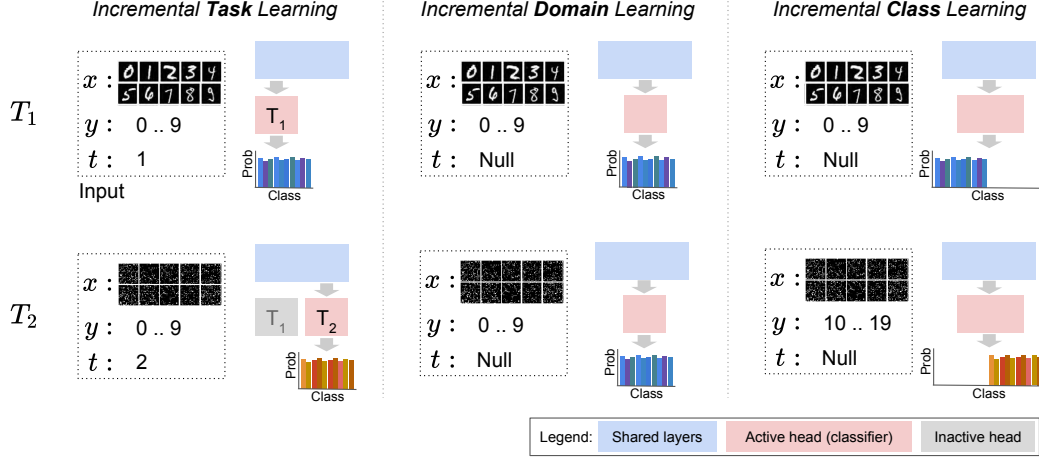## A   Permuted MNIST Experiments

Figure 2: The three continual learning scenarios generated by Permuted MNIST. In each sub-figure, the left dotted rectangle represents the inputs for training, in that (x, y, t) means (input image, target class label, input task identity). The right side illustrates the neural network model and the predicted $P(Y)$ of the model. The color of each bar in the categorical distribution maps to a specific output node in the classifier. Note that the task sequence has 10 different permutations for task T_1 to T_{10} while here we only demonstrate the differences between T_1 and T_2.

In the Permuted MNIST experiments, the pre-processing of an image is similar to Split MNIST except that the order of pixels are permuted. The neural network architecture is also similar to the one in Split MNIST, except that the number of nodes in both hidden layers is 1000. Since the network size is larger, the space overhead introduced in online EWC and SI is larger (#parameter $\approx (1024 \times 1000 + 1000 \times 1000) \times 2 = 4,048,000$) and converts to a buffer of 4k images in Naive rehearsal, or 16k compressed images in Naive rehearsal-C. The generative model in DGR (implemented by [14]) uses a variational autoencoder whose encoder has the same architecture as the classification model; therefore DGR has a similar space overhead as online EWC and SI. In all scenarios, the standard cross-entropy for classification is optimized for 10 epochs with a learning rate ten-times smaller than Split MNIST experiments. Here we use 0.0001 for Adam, and 0.001 for SGD and Adagrad.

The best regularization coefficient for L2, EWC, online EWC, SI, and MAS is obtained through a grid search in each scenario. Our results in Table 3 is averaged from ten runs with random neural network initialization (include randomly initialized parameters in network heads which leads to much better baseline performance, compared to the zero initialization used in [9]). Note that the results of LwF, DGR, and RtF are from [14], which uses the same neural network architecture and training procedure.

Figure 2 illustrated how to use the permutation strategy to generate the three learning scenarios. Note that the number of classes here is larger than the scenarios generated by the splitting strategy (incremental task/domain: 10 versus 2; incremental class: 100 versus 10).

Table 4 compares two different initialization strategies in the hardest scenario, the incremental class learning. The two initialization strategies have the output nodes of all classes pre-allocated or not. In the pre-allocated setting, which has all output nodes be subject to the classification loss right from the beginning, an output node firstly sees negative samples since the first task, followed by some positive samples (in its corresponding task that contains the class), then again sees negative samples in the remaining tasks till the end. In the setting without pre-allocating the output nodes (the default used in our Table 2 and 3 as well as prior works), an output node is created while a new class arrived; thus the output node firstly sees some positive samples (in the newly arrived task), followed by negative

samples in the remaining tasks. The results show that the scenario becomes easier with pre-allocating which enables the output nodes to learn from the beginning of the scenario.

Table 3: The average accuracy (%, higher is better) of all seen tasks after learning with the task sequence generated by Permuted MNIST (Figure 2). The *Memory* column means whether a method uses a memory mechanism, which further divides the methods into two groups in the comparison. The total static memory overhead is controlled to be the same among L2, Naive rehearsal, Naive rehearsal-C, online EWC, SI, MAS, GEM, and DGR. Each value is the average of 10 runs.

	Method	Memory	Incremental task learning	Incremental domain learning	Incremental class learning
Baselines	Adam		93.42 ± 0.56	74.12 ± 0.86	14.02 ± 1.25
	SGD		94.74 ± 0.24	84.56 ± 0.82	12.82 ± 0.95
	Adagrad		94.78 ± 0.18	91.98 ± 0.63	29.09 ± 1.48
	L2		95.45 ± 0.44	91.08 ± 0.72	13.92 ± 1.79
	Naive rehearsal	✓	96.67 ± 0.12	95.19 ± 0.11	96.25 ± 0.10
	Naive rehearsal-C	✓	$\mathbf{97.36} \pm 0.03$	$\mathbf{96.28} \pm 0.47$	$\mathbf{97.24} \pm 0.05$
Continual learning methods	EWC		95.38 ± 0.33	91.04 ± 0.48	26.32 ± 4.32
	Online EWC		95.15 ± 0.49	92.51 ± 0.39	42.58 ± 6.50
	SI		96.31 ± 0.19	93.94 ± 0.45	58.52 ± 4.20
	MAS		96.65 ± 0.18	94.08 ± 0.43	50.81 ± 2.92
	LwF		69.84 ± 0.46	72.64 ± 0.52	22.64 ± 0.23
	GEM	✓	97.05 ± 0.07	96.19 ± 0.11	$\mathbf{96.72} \pm 0.03$
	DGR	✓	92.52 ± 0.08	95.09 ± 0.04	92.19 ± 0.09
	RtF	✓	$\mathbf{97.31} \pm 0.01$	$\mathbf{97.06} \pm 0.02$	96.23 ± 0.04
Offline (upper bound)			98.01 ± 0.04	97.90 ± 0.09	97.95 ± 0.04

Table 4: The average accuracy (%, higher is better) of two **Incremental Class Learning variants**. The first case has an unknown number of total classes (the columns with *No*). The number of output nodes increases along with the total number of seen classes. During training, only the output nodes of seen classes are subject to the classification loss. This is the setting used in Table 2 and 3. The second case has a known number of total classes (the columns with *Yes*). All of the output nodes are pre-allocated and are subject to the classification loss since the first task. The results indicate that the incremental class scenario generated by Permuted MNIST is much easier when the total number of classes is known. In contrast, the scenario generated by Split MNIST has a similar difficulty between the two variants.

	Known Total #class?	Split MNIST		Permuted MNIST	
		No	*Yes*	*No*	*Yes*
Baselines	Adam	19.71 ± 0.08	19.67 ± 0.05	14.02 ± 1.25	42.32 ± 3.22
	SGD	19.46 ± 0.04	19.44 ± 0.03	12.82 ± 0.95	17.54 ± 0.81
	Adagrad	19.82 ± 0.09	19.75 ± 0.08	29.09 ± 1.48	79.50 ± 3.70
	L2	22.52 ± 1.08	20.54 ± 1.12	13.92 ± 1.79	43.18 ± 2.30
	Naive rehearsal	90.78 ± 0.85	89.64 ± 0.63	96.25 ± 0.10	96.24 ± 0.11
	Naive rehearsal-C	95.59 ± 0.49	94.35 ± 0.20	97.24 ± 0.05	97.15 ± 0.05
Continual learning methods	EWC	19.80 ± 0.05	19.76 ± 0.05	26.32 ± 4.32	91.95 ± 1.04
	Online EWC	19.77 ± 0.04	19.71 ± 0.06	42.58 ± 6.50	86.57 ± 3.52
	SI	19.67 ± 0.09	20.88 ± 0.96	58.52 ± 4.20	79.36 ± 2.42
	MAS	19.52 ± 0.29	19.98 ± 0.34	50.81 ± 2.92	73.82 ± 1.67

B Lengthened Task Queue Experiments

Most continual learning methods examine their models with less than 20 tasks in the queue. A more challenging yet practical case of continual learning is to have a dynamic environment where varied
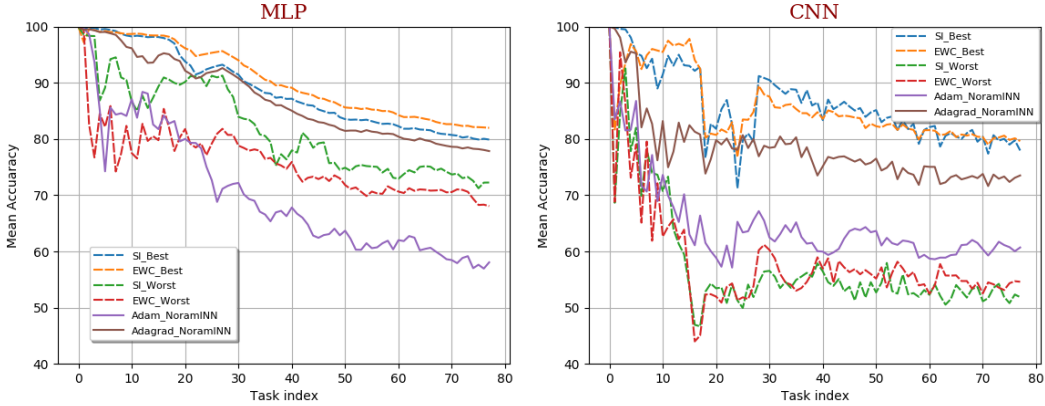
Figure 3: **A comparison between MLP and CNN models**. In each subfigure, we list SI and Online EWC with best and worst hyper-parameter selections (solid line) and two additional optimization methods (dashed line).

Table 5: **Average test accuracy (%) for long task queue with MLP.** Note that SI and Online EWC also apply Adam as optimizer.

	MLP					
Task	Adam	SI (Best)	SI (Worst)	Online EWC (Best)	Online EWC (Worst)	Adagrad
1	99.95	99.95	99.91	99.95	99.81	99.95
11	87.09	98.56	94.77	98.69	92.29	96.14
21	80.01	96.62	90.46	96.78	85.51	92.15
31	72.18	94.06	88.29	94.00	80.81	90.80
41	67.80	88.76	77.46	89.12	72.21	85.03
51	63.68	84.39	71.41	85.61	66.82	81.44
61	62.00	82.48	69.67	84.23	65.76	80.41
71	58.45	81.26	67.04	82.64	62.69	78.61
78	58.05	80.52	66.33	82.06	61.06	77.85

levels of domain shifting are encountered; thus we augment the incremental task learning in Section 3 by extending the length of task queue from 5 to 78 with five datasets, including *MNIST*, *Fashion MNIST*, *EMNIST letter*, *SVHN*, and *CIFAR100*. Each task only contains two classes, while each class presents only once in the task queue.

The evaluation uses two different neural network architectures, CNN and MLP, to enrich the comparison, and we list the detailed architecture in Table 7. For a fair comparison, the number of parameters is similar ($\sim$300K parameters) between CNN and MLP. In the training stage, we adopt Adam as optimizer with a learning rate of 0.001 to train 10 epochs for each task with a batch size of 128. The learning rate for Adagrad is 0.001.

To examine the robustness of the regularization-based methods, we include SI and online EWC in the scenario of longer task queue. The results are presented in Table 5 (MLP) and 6 (CNN), demonstrating that the regularization-based methods can be worse than the baseline (Adam) if the regularization coefficient is not in tuned well. In contrast, the Adagrad achieves a similar level of performance without any hyperparameter search. The sensitivity analysis is provided in Figure 4 and 5, which shows that regularization-based methods are prone to the choice of the regularization coefficient and are sensitive to how the parameters (of the heads) been initialized.

Table 6: **Average test accuracy (%) for long task queue with CNN.** Note that SI and Online EWC also apply Adam as optimizer.

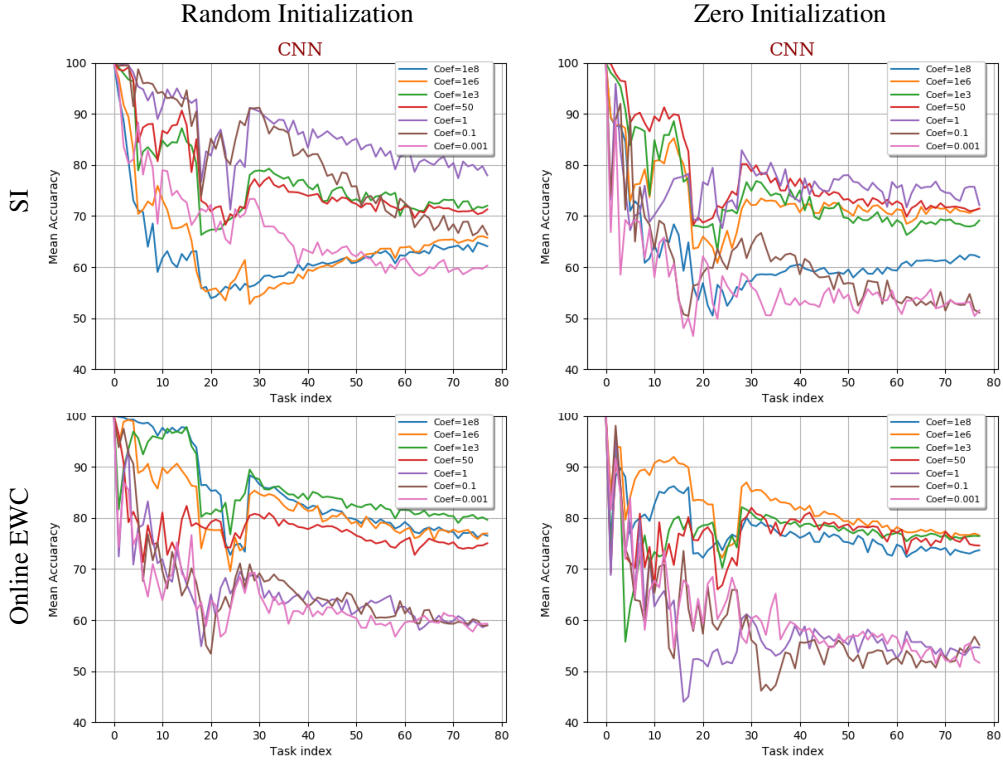| Task	CNN					
	Adam	SI (Best)	SI (Worst)	Online EWC (Best)	Online EWC (Worst)	Adagrad
1	100.0	100.0	100.0	100.0	99.95	100.0
11	73.61	91.46	70.74	95.48	62.65	83.19
21	58.87	81.81	53.42	80.72	51.96	79.80
31	62.47	90.48	56.54	87.54	60.26	78.42
41	59.96	83.35	56.45	83.59	56.94	75.14
51	63.65	85.14	52.70	82.27	55.17	76.51
61	58.72	82.12	52.27	81.58	52.46	75.04
71	60.36	80.40	51.12	80.34	52.52	73.78
78	60.70	77.95	51.87	79.63	54.61	73.51

Figure 4: **Sensitivity to regularization weight.** Top row represents the results of SI, and the bottom row represents the results of Online EWC. Different initialization methods are used in different column.
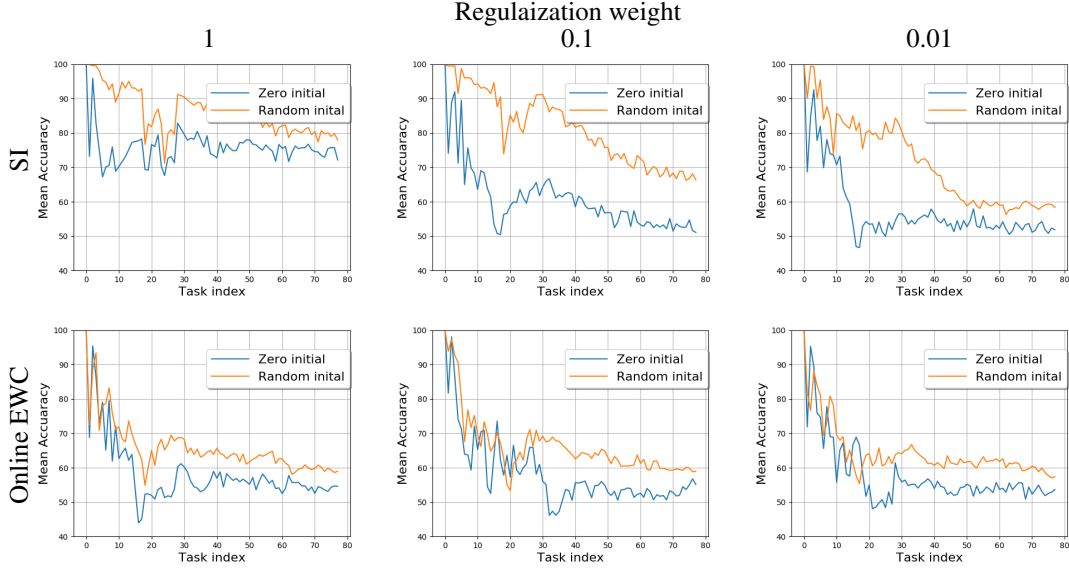
Figure 5: **Sensitivity to initialization method.** Top row represents the results of SI, and the bottom row represents the results of Online EWC. Different regularization weights are used in different columns.

Table 7: The network architecture of CNN and MLP. Note that the input of MLP is a vector flattened from an image of size $28 \times 28 \times 1$.

CNN			
Layer	Activation Size	Activ. Fun.	Max Pooling
Input	$28 \times 28 \times 1$	-	-
$10 \times 5 \times 5$ Conv.	$14 \times 14 \times 10$	ReLU	✓
$20 \times 5 \times 5$ Conv.	$7 \times 7 \times 20$	ReLU	✓
$40 \times 5 \times 5$ Conv.	$3 \times 3 \times 40$	ReLU	-
$70 \times 5 \times 5$ Conv.	$3 \times 3 \times 70$	ReLU	-
Dense Layer	256	ReLU	-
Dense Layer	2	-	-
MLP			
Input	784		
Dense Layer	256	ReLU	-
Dense Layer	256	ReLU	-
Dense Layer	2	-	-