SLOW THINKING ENABLES TASK-UNCERTAIN LIFELONG AND SEQUENTIAL FEW-SHOT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Lifelong machine learning focuses on adapting to novel tasks without forgetting the old tasks, whereas few-shot learning strives to learn a single task given a small amount of data. These two different research areas are crucial for artificial general intelligence, however, their existing studies have somehow assumed some impractical settings when training the models. For lifelong learning, the nature (or the quantity) of incoming tasks during inference time is assumed to be known at training time. As for few-shot learning, it is commonly assumed that a large number of tasks is available during training. Humans, on the other hand, can perform these learning tasks without regard to the aforementioned assumptions. Inspired by how the human brain works, we propose a novel model, called the Slow Thinking to Learn (STL), that makes sophisticated (and slightly slower) predictions by iteratively considering interactions between current and previously seen tasks at runtime. Having conducted experiments, the results empirically demonstrate the effectiveness of STL for more realistic lifelong and few-shot learning settings.

1 INTRODUCTION

Deep Learning has been successful in various applications. However, it still has a lot of areas to improve on to reach human's lifelong learning ability. As one of its drawbacks, neural networks (NNs) need to be trained on large datasets before giving satisfactory performance. Additionally, they usually suffer from the problem of catastrophic forgetting (McCloskey & Cohen (1989); French (1999))—a neural network performs poorly on old tasks after learning a novel task. In contrast, humans are able to incorporate new knowledge even from few examples, and continually throughout much of their lifetime. To bridge this gap between machine and human abilities, effort has been made to study few-shot learning (Fei-Fei et al. (2006); Lake et al. (2011); Santoro et al. (2016); Vinyals et al. (2016); Snell et al. (2017); Ravi & Larochelle (2017b); Finn et al. (2017); Sung et al. (2018); Garcia & Bruna (2018); Qi et al. (2018)), lifelong learning (Gepperth & Karaoguz (2016); Rusu et al. (2016); Kirkpatrick et al. (2017); Yoon et al. (2018); Kemker & Kanan (2018); Kemker et al. (2018); SerrÃ et al. (2018); Schwarz et al. (2018); Sprechmann et al. (2018); Riemer et al. (2019)), and both (Kaiser et al. (2017)).

The learning tasks performed by humans are, however, more complicated than the settings used by existing lifelong and few-shot learning works. **Task uncertainty**: currently, lifelong learning models are usually trained with hyperparameters (e.g., number of model weights) optimized for a sequence of tasks arriving *at test time*. The knowledge about future tasks (even their quantity) may be a too strong assumption in many real-world applications, yet without this knowledge, it is hard to decide the appropriate model architecture and capacity when training the models. **Sequential few-shot tasks**: existing few-shot learning models are usually (meta-)trained using a *large* collection of tasks.[1] Unfortunately, this collection is not available in the lifelong learning scenarios where tasks come in sequentially. Without

[1]In Finn et al. (2017); Kaiser et al. (2017), the number of available tasks during a few-shot training can be as large as $P_5^{1200} \approx 2e12$ given the Omniglot dataset, 1200 training characters, and 5-way classification.

seeing many tasks at training time, it is hard for an existing few-shot model to learn the shared knowledge behind the tasks and use the knowledge to speed up the learning of a novel task at test time.

Humans, on the other hand, are capable of learning well despite having only limited information and/or even when not purposely preparing for a particular set of future tasks. Comparing how humans learn and think to how the current machine learning models are trained to learn and make predictions, we observe that the key difference lies on the part of *thinking*, which is the decision-making counterpart of models when making predictions. While most NN-based supervised learning models use a single forward pass to predict, humans make careful and less error-prone decisions in a more sophisticated manner. Studies in biology, psychology, and economics (Parisi et al. (2019); Kahneman & Egan (2011)) have shown that, while humans make fast predictions (like machines) when dealing with daily familiar tasks, they tend to rely on a slow-thinking system that deliberately and iteratively considers interactions between current and previously learned knowledge in order to make correct decisions when facing unfamiliar or uncertain tasks. We hypothesize that this slow, effortful, and less error-prone decision-making process can help bridge the gap of learning abilities between humans and machines.

We propose a novel brain-inspired model, called the Slow Thinking to Learn (STL), for task-uncertain lifelong and sequential few-shot machine learning tasks. STL has two specialized but dependent modules, the cross-task Slow Predictor (SP) and per-task Fast Learners (FLs), that output lifelong and few-shot predictions, respectively. We show that, by making the prediction process of SP more sophisticated (and slightly slower) at runtime, the learning process of all modules can be made easy at training time, eliminating the need to fulfill the aforementioned impractical settings. Note that the techniques for slow predictions (Finn et al. (2017); Ravi & Larochelle (2017b); Nichol & Schulman (2018); Sprechmann et al. (2018)) and fast learning (McClelland et al. (1995); Kumaran et al. (2016); Kaiser et al. (2017)) have already been proposed in the literature. Our contributions lie in that we 1) explicitly model and study the *interactions* between these two techniques, and 2) demonstrate, for the first time, how such interactions can greatly improve machine capability to solve the joint lifelong and few-shot learning problems encountered by humans everyday.

2 Slow Thinking to Learn (STL)

We focus on a practical lifelong and few-shot learning set-up:

Problem 1. Given tasks $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \cdots$ arriving in sequence and the labeled examples $\mathcal{D}^{(t)} = (\mathcal{X}^{(t)}, \mathcal{Y}^{(t)}) = \{(\boldsymbol{x}^{(t,i)}, \boldsymbol{y}^{(t,i)})\}_t$ in each task $\mathcal{T}^{(t)}$ also coming in sequence, the goal is to design a model such that it can be properly trained by data $\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \cdots, \mathcal{D}^{(s)}$ collected up to any given time point s, and then make correct predictions for unlabeled data $\mathcal{X}'^{(t)} = \{\boldsymbol{x}'^{(t,i)}\}_i$ in any of the seen tasks, $t \leq s$.

Note that, at training time s, the future tasks $\mathcal{T}^{(s+1)}, \mathcal{T}^{(s+2)}, \cdots$ are unknown, and the training set $\mathcal{D}^{(s)}$ of the last task $\mathcal{T}^{(s)}$ may consist of only few examples. The model should not assume any knowledge from $\mathcal{T}^{(s+1)}, \mathcal{T}^{(s+2)}, \cdots$ and should learn from few shots in $\mathcal{D}^{(s)}$ even when s is small.

To solve Problem 1, we propose the Slow Thinking to Learn (STL) model, whose architecture is shown in Figure 1. The STL

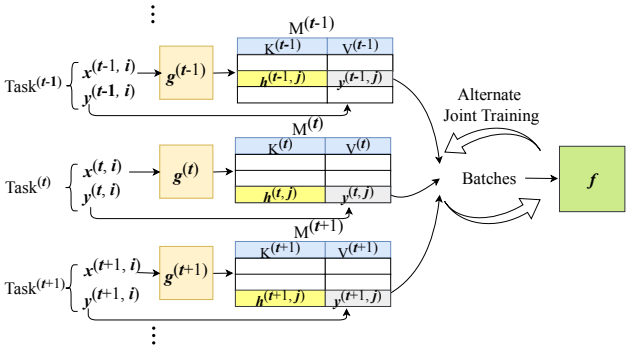


Figure 1: The Slow Thinking to Learn (STL) model. To model the interactions between the shared SP f and per-task FLs $\{(g^{(t)}, \mathcal{M}^{(t)})\}_t$, we feed the output of FLs into the SP while simultaneously letting the FLs learn from the feedback given by SP.

is a cascade where the shared Slow Predictor (SP) network f parameterized by θ takes the output of multiple task-specific Fast Learners (FLs) $\{(g^{(t)}, \mathcal{M}^{(t)})\}_t$, $t \leq s$, as input. An FL for task $\mathcal{T}^{(t)}$ consists of an embedding network $g^{(t)}$[2] parameterized by $\phi^{(t)}$ and augmented with an external, episodic, non-parametric memory $\mathcal{M}^{(t)} = (\mathcal{K}^{(t)}, \mathcal{V}^{(t)}) = \{(\boldsymbol{h}^{(t,j)}, \boldsymbol{v}^{(t,j)})\}_j$. Here, we use the Memory Module (Kaiser et al. (2017)) as the external memory which saves the *clusters* of seen examples $\{(\boldsymbol{x}^{(t,i)}, \boldsymbol{y}^{(t,i)})\}_i$ to achieve better storage efficiency—the $\boldsymbol{h}^{(t,j)}$ of an entry $(\boldsymbol{h}^{(t,j)}, \boldsymbol{v}^{(t,j)})$ denotes the embedding of a cluster of $\boldsymbol{x}^{(t,i)}$'s with the same label while the $\boldsymbol{v}^{(t,j)}$ denotes the shared label.

We use the FL $(g^{(t)}, \mathcal{M}^{(t)})$ and SP f to make few-shot and lifelong predictions for task $\mathcal{T}^{(t)}$, respectively. We let the number of FLs grow with the number of seen tasks in order to ensure that the entire STL model will have enough complexity to learn from possibly endless tasks in lifelong. This does *not* imply that the SP will consume unbounded memory space to make predictions at runtime, as the FL for a specific task can be stored on a hard disk and loaded into the main memory only when necessary.

Slow Predictions. The FL predicts the label of a test instance $\boldsymbol{x}'$ using a single feed-forward pass just like most existing machine learning models. As shown in Figure 2(a), the FL for task $\mathcal{T}^{(t)}$ first embed the instance to get $\boldsymbol{h}' = g^{(t)}(\boldsymbol{x}')$ and then predicts the label $\hat{\boldsymbol{y}}'_{\text{FL}}$ of $\boldsymbol{x}'$ by averaging the cluster labels $\boldsymbol{v}^{(t,j)}$'s stored in $\mathcal{M}^{(t)}$ whose corresponding representations $\boldsymbol{h}^{(t,j)}$'s are most similar to $\boldsymbol{h}'$. Specifically, let

$$\hat{\mathcal{M}}^{(t)}_{\boldsymbol{h}'} = \{(\boldsymbol{h}, \boldsymbol{v}) \in \mathcal{M}^{(t)} : \boldsymbol{h} \in \text{KNN}(\boldsymbol{h}')\},$$

where $\text{KNN}(\boldsymbol{h}')$ is the set of K nearest neighboring embeddings of $\boldsymbol{h}'$. We have

$$\hat{\boldsymbol{y}}'_{\text{FL}} = \textstyle\sum_{(\boldsymbol{h},\boldsymbol{v}) \in \hat{\mathcal{M}}^{(t)}_{\boldsymbol{h}'}} \langle \boldsymbol{h}, \boldsymbol{h}' \rangle \, \boldsymbol{v},$$

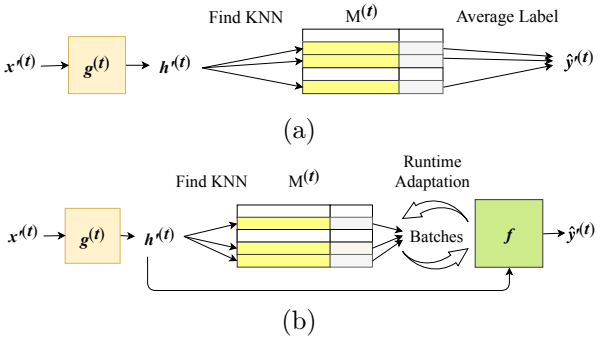


Figure 2: The prediction processes of (a) an FL, and (b) the SP at runtime.

where $\langle \boldsymbol{h}, \boldsymbol{h}' \rangle$ denotes the cosine similarity between $\boldsymbol{h}^{(t,j)}$ and $\boldsymbol{h}'$. On the other hand, the SP predicts the label of $\boldsymbol{x}'$ with a slower, iterative process, which is shown in Figure 2(b). The SP first adapts (i.e., fine-tunes) its weights θ to $\text{KNN}(\boldsymbol{h}')$ and their corresponding values stored in $\mathcal{M}^{(t)}$ to get $\tilde{\theta}'$ by solving

$$\tilde{\theta}' = \arg_{\tilde{\theta}} \min \textstyle\sum_{(\boldsymbol{h},\boldsymbol{v}) \in \hat{\mathcal{M}}^{(t)}_{\boldsymbol{h}'}} \text{loss}(f(\boldsymbol{h}; \tilde{\theta}), \boldsymbol{v}) \text{ subject to } \|\tilde{\theta} - \theta\| \leq R, \tag{1}$$

where $\text{loss}(\cdot)$ denotes a loss function. Then, the SP makes a prediction by $\hat{\boldsymbol{y}}'_{\text{SP}} = f(\boldsymbol{h}'; \tilde{\theta}')$. The adapted network $f_{\tilde{\theta}'}$ is discarded after making the prediction.

The slower decision-making process of SP may seem unnecessary and wasteful of computing resources at first glance. Next, we explain why it is actually a good bargain.

Life-Long Learning with Task Uncertainty. Since the SP makes predictions after runtime adaptation, we define the training objective of θ for task $\mathcal{T}^{(s)}$ such that it minimizes the losses *after* being adapted for each seen task $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \cdots, \mathcal{T}^{(s)}$:

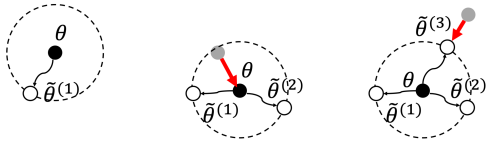
$$\begin{aligned} &\arg_{\theta} \min \textstyle\sum_{t:t \leq s} \mathcal{L}_{\text{SP}}(\theta; g^{(t)}, \mathcal{M}^{(t)}), \text{ where} \\ &\mathcal{L}_{\text{SP}}(\theta; g^{(t)}, \mathcal{M}^{(t)}) = \textstyle\sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{M}^{(t)}} \text{loss}(f(\boldsymbol{x}; \tilde{\theta}^*), \boldsymbol{y}). \end{aligned} \tag{2}$$

The term $\text{loss}(f(\boldsymbol{h}; \tilde{\theta}^*), \boldsymbol{v})$ denotes the empirical slow-prediction loss of the adapted SP on an example $(\boldsymbol{x}, \boldsymbol{y})$ in $\mathcal{M}^{(t)}$, where $\tilde{\theta}^*$ denotes the weights of the adapted SP for $\boldsymbol{x}$ following Eq. (1): $\tilde{\theta}^* = \arg_{\tilde{\theta}} \min \sum_{(\boldsymbol{h},\boldsymbol{v}) \in \hat{\mathcal{M}}^{(t)}_{\boldsymbol{x}}} \text{loss}(f(\boldsymbol{h}; \tilde{\theta}), \boldsymbol{v})$ subject to $\|\tilde{\theta} - \theta\| \leq R$. Solving Eq. (2)

[2]In practice, different FLs can have either their own embedding networks $g^{(t)}$'s or a shared one, depending on whether the corresponding tasks are coming from distinct domains or not.

requires recursively solving $\tilde{\theta}^*$ for each $(\boldsymbol{x}, \boldsymbol{y})$ remembered by the FLs. We use an efficient gradient-based approach proposed by Finn et al. (2017)) to solve Eq. (2). Please refer to Section 2.1 of the Appendix for more details. Since the SP learns from the output of FLs, the $\tilde{\theta}^*$ in Eq. (2) approximates a hypothesis used by an FL to predict the label of $\boldsymbol{x}$. The θ, after being trained, will be close to every $\tilde{\theta}^*$ and can be fine-tuned to become a hypothesis, meaning that θ encodes the *invariant principles*[3] underlying the hypotheses for different tasks.

Recall that in Problem 1, the nature of tasks arriving after a training process is unknown, thus, it is hard to decide the right model capacity at training time. A solution to this problem is to use an expandable network (Rusu et al. (2016); Yoon et al. (2018)) and expand the network when training it for a new task, but the number of units to add during each expansion remains unclear. Our STL walks around this problem by *not* letting the SP learn the tasks directly but making it learn the invariant principles behind the tasks. Assuming that the underlying principles of the learned hypotheses for different tasks are universal and relatively simple,[4] one only needs to choose a model architecture with capacity that is enough to learn the shared principles in lifelong manner. Note that limiting the capacity of SP at training time does not imply underfitting. As shown in Figure 3, the post-adaptation capacity of SP at runtime can be much larger than the capacity decided during training.



(a) (b) (c)

Figure 3: The relative positions between the invariant representations θ and the approximate hypotheses $\tilde{\theta}^{(t)}$'s of FLs for different tasks $\mathcal{T}^{(t)}$'s on the loss surface defined by FLs after seeing the (a) first, (b) second, and (c) third task. Since $\|\theta - \tilde{\theta}^{(t)}\| \leq R$ for any t in Eq. (2), the effective capacity of SP (at runtime) is the union of the capacity of all possible points within the dashed R-circle centered at θ. Furthermore, after being sequentially trained by two tasks using Eq. (3), the θ will easily get stuck in the middle of $\tilde{\theta}^{(1)}$ and $\tilde{\theta}^{(2)}$. To solve the third task, the third FL needs to change its embedding function (and therefore the loss surface) such that $\tilde{\theta}^{(3)}$ falls into the R-circle centered at θ.

Sequential Few-Shot Learning. Although each FL is augmented with an external memory that has been shown to improve learning efficiency by the theory of complementary learning systems (McClelland et al. (1995); Kumaran et al. (2016)), it is not sufficient for FLs to perform few-shot predictions. Normally, these models need to be trained on many existing few-shot tasks in order to obtain good performance at test time. Without assuming s in Problem 1 to be a large number, the STL takes a different approach that fast stabilizes θ and then let the FL for a new incoming task learn a good hypothesis by extrapolating from θ. We define the training objective of $g^{(s)}$, which is parameterized by $\phi^{(s)}$ and augmented with memory $\mathcal{M}^{(s)}$, for the current task $\mathcal{T}^{(s)}$ as follows:

$$\arg_{\phi^{(s)}} \min \mathcal{L}_{\text{FL}}(\phi^{(s)}; \mathcal{D}^{(s)}, \mathcal{M}^{(s)}) + \lambda \Omega(\phi^{(s)}; \theta, \mathcal{D}^{(s)}, \mathcal{M}^{(s)}), \tag{3}$$

where $\mathcal{L}_{\text{FL}}(\phi^{(s)}; \mathcal{D}^{(s)}, \mathcal{M}^{(s)})$ is the empirical loss term whose specific form depends on the type of external memory used (see Section 2.2 of the Appendix for more details), and $\Omega(\phi^{(s)}; \theta, \mathcal{D}^{(s)}, \mathcal{M}^{(s)})$ is a regularization term, which we call the *feedback term*, whose inverse value denotes the usefulness of the FL in helping SP (f parameterized by θ) adapt. Specifically, it is written as

$$\Omega(\phi^{(s)}; \theta, \mathcal{D}^{(s)}, \mathcal{M}^{(s)}) = \textstyle\sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}^{(s)}} \text{loss}(f(\boldsymbol{x}; \tilde{\theta}^*), \boldsymbol{y}), \text{ where}$$
$$\tilde{\theta}^* = \arg_{\tilde{\theta}} \min \textstyle\sum_{(\boldsymbol{h},\boldsymbol{v}) \in \hat{\mathcal{M}}^{(s)}_{g^{(s)}(\boldsymbol{x}; \phi^{(s)})}} \text{loss}(f(\boldsymbol{h}; \tilde{\theta}), \boldsymbol{v}) \text{ subject to } \|\tilde{\theta} - \theta\| \leq R.$$

The feedback term encourages each FL to learn unique and salient features for the respective task so the SP will not be confused by two tasks having similar embeddings. As shown in

[3]The word "invariant" does not imply causal invariance in the field of causal learning.

[4]For example, in Physics, the principles behind many complex systems are usually governed by few basic and fundamental physical laws.

Figure 3(b), the relative position of θ gets "stuck" easily after seeing a few of previous tasks. To solve the current task, $g^{(s)}$ needs to change the loss surface for θ such that $\tilde{\theta}^{(s)}$ falls into the R-circle centered at θ (Figure 3(c)). This makes θ an efficient guide (through the feedback term) to finding $g^{(s)}$ when there are only few examples and also few previous tasks.

We use an alternate training procedure to train the SP and FLs. Please see Section 2.3 of the Appendix for more details. Note that when sequentially training STL for task $\mathcal{T}^{(s)}$ in lifelong, we can safely discard the data $\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \cdots, \mathcal{D}^{(s-1)}$ in the previous tasks because the FLs are task-specific (see Eq. (3)) and the SP does not require raw examples to train (see Eq. (2)).

3 Further Related Work

In this section, we discuss related works that are not mentioned in Sections 1 and 2. For a complete discussion, please refer to Section 1 of the Appendix.

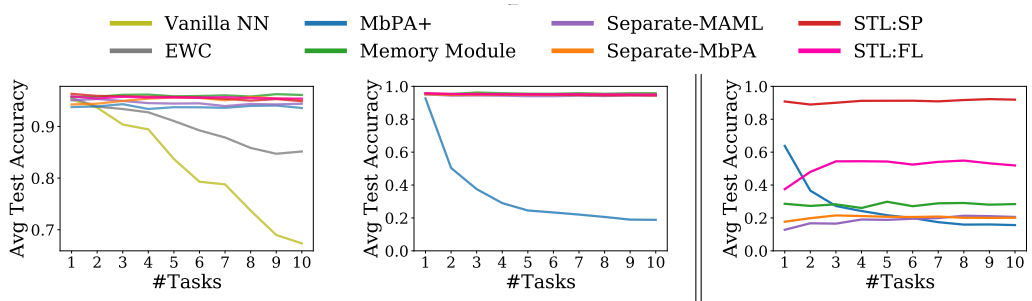
Runtime Adaptation. Our idea of adapting SP at runtime is similar to that of MbPA (Sprechmann et al. (2018)), which is a method proposed for lifelong learning only. In MbPA, the embedder and output networks are trained together, in a traditional approach, as one network for the current task. Its output network adapts to examples stored in an external memory for a previous task before making lifelong predictions. Nevertheless, there is no discussion of how the runtime adaptation could improve the learning ability of a model, which is the main focus of this paper. **Meta-Learning.** The idea of learning the invariant representations in SP is similar to meta-learning (Finn et al. (2017); Ravi & Larochelle (2017b); Nichol & Schulman (2018)), where a model (meta-)learns good initial weights that can speed up the training of the model for a new task using possibly only few shots of data. To learn the initial weights (which correspond to the invariant representations in our work), existing studies usually assume that the model can sample tasks, including training data, following the task distribution of the ground truth. However, the Problem 1 studied in this paper does not provide such a luxury. **Memory-Augmented Networks.** An FL is equipped with an external episodic memory module, which is shown to have fast-learning capability (McClelland et al. (1995); Kumaran et al. (2016)) due to its nonparametric nature. Although we use the Memory Module (Kaiser et al. (2017)) in this work, our model can integrate with other types of external memory modules, such as Gepperth & Karaoguz (2016); Pritzel et al. (2017); Santoro et al. (2016). This is left as our future work. **Few-Shot Learning without Forgetting.** Recently, Gidaris & Komodakis (2018) proposed a new few-shot learning approach that does not forget previous tasks when trained on a new one. However, it still needs to be trained on a large number of existing tasks in order to make few-shot predictions and therefore cannot be applied to Problem 1.

4 Experimental Evaluation

In this section, we evaluate our model in different aspects.

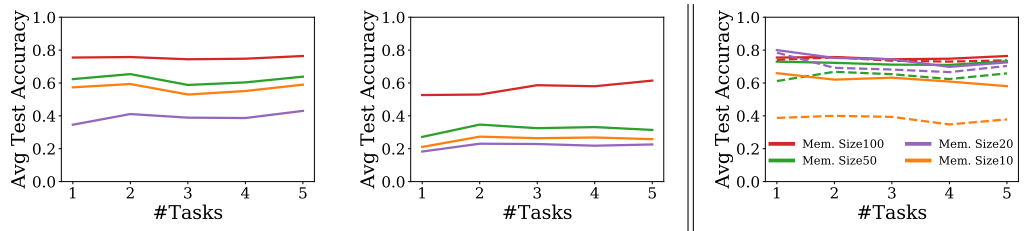
4.1 Task-Uncertain Lifelong Learning

We implement STL and the following baselines using TensorFlow (Abadi et al. (2016)): **Vanilla NN.** A neural network without any technique for preventing catastrophic forgetting or preparation for few-shot tasks. **EWC.** A regularization technique (Kirkpatrick et al. (2017)) protecting the weights that are important to previous tasks in order to mitigate catastrophic forgetting. **Memory Module.** An external memory module (Kaiser et al. (2017)) that can make predictions (using KNNs) by itself. It learns to cluster rows to improve prediction accuracy and space efficiency. **MbPA+.** A memory-augmented model (Sprechmann et al. (2018)) that performs runtime adaptation like our SP before making lifelong predictions. The original MbPA uses a FIFO memory, which we replace with the Memory Module for better performance because of its clustering technique. **Separate-MAML.** A cascade model with the same architecture as STL, except that there is no feedback term in Eq. (3). Without the feedback term, the FLs and SP can be separately



(a) Mem. size: 1000 (Raw) (b) Mem. size: 1000 (Emb.) (c) Mem. size: 10 (Emb.)

Figure 4: Average all-task performance after seeing different number of sequential tasks on the permuted MNIST dataset. For memory-augmented networks, we store at most (a) 1000 raw examples, (b) 1000 embedded examples, and (c) 10 embedded examples in their external memory.



(a) Normal (b) Hard (c) STL on Normal

Figure 5: Average all-task performance after seeing different number of sequential (a) CIFAR-100 Normal tasks and (b) CIFAR-100 Hard tasks. The size of external memory is set to 100 embedded examples. Colors follow Figure 4. (c) Performance of SP with different memory sizes. Dashed lines show the average performance of FLs.

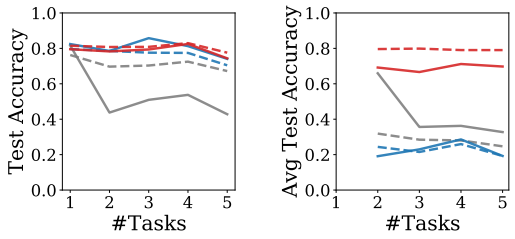
trained (FLs first, and then SP), and we use MAML (Finn et al. (2017)) to solve Eq. (2) when training the SP. **Separate-MbPA.** This is similar to Separate-MAML, except that the SP is not trained to prepare for run-time adaptation, but it still applies run-time adaptation at test time.

Next, we evaluate the abilities of the models to fight against catastrophic forgetting using the permuted MNIST (LeCun et al. (1998)) and CIFAR-100 (Krizhevsky & Hinton (2009)) datasets. Then, we investigate the impact of task-uncertainty on model performance.

Permuted MNIST. We create a sequence of 10 tasks, where each task contains MNIST images whose pixels are randomly permuted using the same seed. The seeds are different across tasks. We train models for one task at a time, and test their performance on all tasks seen so far. We first use a setting where all memory-augmented models can save raw examples in their external memory. This eliminates the need for an embedding network, and, following the settings in Kirkpatrick et al. (2017), we use a 2-layer MLP with 400 units for all models. We trained all models using the Adam optimizer for 10,000 iterations per task, with their best-tuned hyperparameters. Figure 4(a) shows the average performance of models for all tasks seen so far. The memory-augmented models outperform the Vanilla NN and EWC and do not suffer from forgetting. This is consistent with previous findings (Sprechmann et al. (2018); Kaiser et al. (2017)). However, saving raw examples for a potentially infinite number of tasks may be infeasible as it consumes a lot of space. We therefore use another setting where memory-augmented models save only the embedded examples. This time, we let both the embedder and the output network (in STL, it is SP) consist of 1-layer MLP with 400 units. Figure 4(b) shows that the memory-augmented models do not forget even when saving the embeddings. The only exception is MbPA+, because it uses the same embedder network for all tasks, the embedder network is prone to forgetting.

CIFAR-100. Here, we design more difficult lifelong learning tasks using the CIFAR-100 dataset. The CIFAR-100 dataset consists of 100 classes. Each class belongs to a superclass, and each superclass has 5 classes. We create a sequence of tasks, called CIFAR-100 Normal, where the class labels in one task belong to different superclasses, but the labels across different tasks are from the same superclass. This ensures that there is transferable knowledge between tasks in the ground truth. We also create another sequence of tasks, called CIFAR-100 Hard, where the class labels in one task belong to the same superclasses, while the labels across different tasks are from different superclass. The tasks in CIFAR-100 Hard share less information, making the lifelong learning more difficult. For CIFAR-100 tasks, we let the memory-augmented models store embeddings in external memory. The embedding networks of all models consist of 4 convolutional layers followed by one fully connected layer, and all output networks (SP in STL) are a 2-layer MLP with 400 units. We search for the best hyperparameters for each model but limit the memory size to 100 embeddings, apply early stopping during training, and use Adam as the optimizer. As shown in Figures 5(a)(b), our SP clearly outperforms the baseline models for both the Normal and Hard tasks.

Task Uncertainty and Hyperparameters. To understand why the SP outperforms other baselines, we study how the performance of each model changes with model capacity. Figure 4(c) shows the performance of different models on the permuted MNIST dataset when we deliberately limit the size of external memory to 10 embeddings. Only the SP performs well in this case. We also vary the size of external memory used by our FLs and find out that the performance of SP does not drastically change like the other baselines, except when the memory size is extremely small, as shown in Figure 5(c). The above results justify that our STL can avoid the customization of memory size (a hyperparameter) to be specifically catered to expected future tasks, whose precise characteristics may not be known at training time. In addition to memory size, we also conduct experiments with models whose architectures of the output networks (SP in our STL) are changed based on LeNet (LeCun et al. (1998)). We consider two model capacities. The larger model has 4 convolutional layers with 128, 128, 256, and 256 filters followed by 3 fully-connected layers with 256, 256, and 128 units; whereas the small model has 4 convolutional layers with 16, 16, 32, and 32 filters followed by 3 fully-connected layers with 64, 32, 16 units. Figure 6 compares the performance of different parametric models for the current and previous CIFAR-100 Normal tasks. We can see that the performance of EWC on current task is heavily affected by model capacity. EWC with small model size can learn well at first, but struggles to fit the following tasks, which was not a problem when it has larger model size. MbPA has good performance on current task but forgets the previous tasks no matter how large the model is. On the other hand, STL is able to perform well on both the previous and current tasks regardless of model size. This proves the advantage of SP's runtime adaptation ability, that is, it mitigates the need for a model that is carefully sized to the incoming uncertain lifelong tasks.

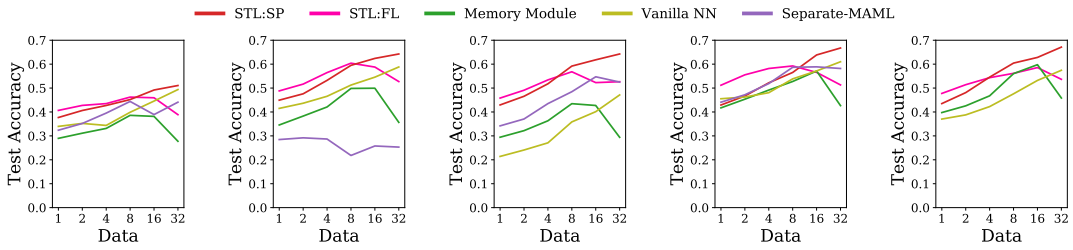


(a) Latest Task (b) Previous Tasks

Figure 6: Performance for (a) the latest task and (b) all previous tasks after seeing different number of sequential CIFAR-100 Normal tasks. Colors follow Figure 4. Solid and dashed lines denote models with large and small capacities, respectively.

4.2 Sequential Few-shot Learning

CIFAR-100. Existing few-shot learning models are usually trained using a large collection of tasks as training batches. However, in sequential continual learning settings, collections of these tasks are not available. Here we designed an experiment setting that simulates an incoming few-shot task during lifelong learning. We modified the CIFAR Normal and CIFAR Hard sequential tasks, where we trained the models with sequential tasks just like



(a) 2-nd Task (b) 4-th Task (c) 6-th Task (d) 8-th Task (e)10-th Task

Figure 7: Performance for the (a) 2-nd, (b) 4-th, (c) 6-th, (d) 8-th, and (e) 10-th sequential CIFAR-100 Normal task that has only a few shots (in number of batches) of training data.

conventional lifelong learning set-up, except that the last task is a "few-shot" task.[5] In this experiment, we assume that the input domains are the same, which means we can use the network parameters (e.g. embedder's weights) learned from previous tasks as initial weights. We consider three baselines, namely the Memory Module, Separate-MAML, and Vanilla NN. The Memory Module is the only known model designed for both lifelong and few-shot learning. We use Separate-MAML to simulate the STL without the feedback term in Eq (3) and the Vanilla NN to indicate "default" fine-tuning performance for each task. Figure 7 shows the performance on the few-shot task with different number of batches of available training data. Each batch contains 16 examples, and the memory size for each task is 20 in all memory-augmented models. We can see that both the FLs and SP in our model outperform other baselines. The Memory Module cannot learn well without seeing a large collection of tasks at training time, and the Separate-MAML gives unstable performance due to the lack of feedback from the SP (MAML). Interestingly, these two sophisticated models perform worse than the Vanilla NN sometimes, justifying that the interactions between the fast-leaning and slow-thinking modules are crucial to the joint lifelong and few-shot learning. Our above observations still hold on the even more challenging dataset, CIFAR Hard. Please refer to Section 3.3 of the Appendix for more details.

Comparing the results of FLs and SP, we can see that an FL gives better performance when the training data is small. This justifies that the invariant representations learned by the SP can indeed guide an FL to better learn from the few shots. Interestingly, the intersection of the predictive ability of an FL and the SP seem to be stable across tasks and usually falls within the range of 48 to 192 examples. In Section 3.4 of the Appendix, we visualize the embeddings stored in the FLs and the Memory Module to understand how the feedback from SP guide the representation learning of FLs.

4.3 Time and Space Consumption

Inference Time. The SP makes "slow" predictions because of runtime adaptation. Here, we study the time required by the SP to make a single prediction. We run trained models on a machine with a commodity NVIDIA GTX-1070 GPU. The number of adaptation steps used is 3 as in previous experiments. For an FL, SP, and a non-adaptive Vanilla NN trained for the CIFAR-100 Normal tasks, we get 0.24 ms, 2.62 ms, and 0.79 ms per-example inference time on average. We believe that trading delay of a few milliseconds at runtime for a great improvement on lifelong and few-shot learning abilities is a good bargain in many applications. **Space Efficiency.** The STL also has an advantage in space efficiency. Please see Section 3 of the Appendix for more details.

[5]For clarity, our few-shot task is different from the settings of Finn et al. (2017); Vinyals et al. (2016) in which they have an example for each class in a shot, however, our settings is "batch based." That is, we randomly sampled few batches as our training data, thus, the class distribution may not be uniform.

5 Conclusion

Inspired by the *thinking* process that humans undergo when making decisions, we propose STL, a cascade of per-task FLs and shared SP. To the best of our knowledge, this is the first work that studies the interactions between the fast-learning and slow-prediction techniques and shows how such interactions can greatly improve machine capability to solve the joint lifelong and few-shot learning problems under challenging settings. For future works, we will focus on integrating the STL with different types of external memory and studying the performance of STL in real-world deployments.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. 2016.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 28(4):594–611, 2006.

Chelsea Finn, Pieter Abbeel, and Sergy Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. of ICML*, 2017.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. In *Proc. of ICLR*, 2018.

Alexander Gepperth and Cem Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934, 2016.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4367–4375, 2018.

Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. In *Proc. of ICLR*, 2018.

Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. Overcoming catastrophic forgetting via model adaptation. In *Proc. of ICLR*, 2019. URL `https://openreview.net/forum?id=ryGvcoA5YX`.

Daniel Kahneman and Patrick Egan. *Thinking, fast and slow*. Farrar, Straus and Giroux New York, 2011.

Lukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. In *Proc. of ICLR*, 2017.

Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. international conference on learning representations. In *Proc. of ICLR*, 2018.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proc. of AAAI*, 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness Guillaume Desjardins, Andrei Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. In *Proc. of National Academy of Sciences (PNAS)*, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. In *Proc. of the Annual Conf. of the Cognitive Science Society*, 2011.

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

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Proc. of NIPS*, 2017.

Zhizhong Li and Derek Hoiem. Learning without forgetting. In *Proc. of ECCV*, 2016.

David Lopez-Paz and Marc' Aurelio Ranzato. Gradient episodic memory for continual learning. In *Proc. of NIPS*, 2017.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

James McClelland, Bruce McNaughton, and Randall O'Reilly. 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*, 1995.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 1989.

Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

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

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *Proc. of ICML*, 2017.

Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proc. of CVPR*, 2018.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *Proc. of ICLR*, 2017a.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *Proc. of ICLR*, 2017b.

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

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1802.07569*, 2016.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proc. of ICML*, 2016.

Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *Proc. of ICML*, 2018.

Joan SerrÃ , DÃdac SurÃs, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *Proc. of ICML*, 2018.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Proc. of NIPS*, 2017.

Pablo Sprechmann, Siddhant Jayakumar, Jack Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uri, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. Memory-based parameter adaptation. In *Proc. of ICLR*, 2018.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proc. of CVPR*, 2018.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Proc. of NIPS*, 2016.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *Proc. of ICLR*, 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proc. of ICML*, 2017.

Appendix

1 Related Work

In this section, we give a more thorough review of the related work.

1.1 Runtime Adaptation

MbPA (Sprechmann et al. (2018)), a memory-augmented network, proposed an *adaptive* manner of utilizing its memory contents. Its output network applies a run-time adaptation using the stored information from its external non-parametric first-in first-out (FIFO) memory. It was able to show that such method can help the output network to deal with *catastrophic forgetting* – the problem of machine learning models when after learning a new task, they would perform poorly on old tasks. However, MbPA did not focus on how such a run-time adaptation can help the entire model (including the memory) to learn better, which is the main focus of our proposed Slow Thinking to Learn (STL) model.

1.2 Meta-Learning and Few-Shot Learning

Some of the most notable Meta-Learning research for few-shot learning are the following: meta-LSTM (Ravi & Larochelle (2017a)), MAML (Finn et al. (2017)), and Reptile (Nichol & Schulman (2018)). Meta-LSTM proposes a meta-learner LSTM that controls the gradient updates of another network. On the other hand, MAML and Reptile aim at learning an easily adaptable set of weights, wherein the weights are updated using the gradient, rather than a learned update. Our idea of learning the invariant representations across tasks is similar to these meta-learning models. The difference is that all of the aforementioned previous works solve the few-shot learning problem by sampling a large number of few-shot tasks from a meta-distribution. However, this assumption of having a large amount of few-shot training tasks, is not practical and is not available in the sequential few-shot learning problem that our model would like to solve. Recently, Gidaris & Komodakis (2018) studied the problem of few-shot learning without forgetting. Their idea of extracting and transferring useful general knowledge from previous tasks to a new, unseen few-shot task is similar to ours. However, the proposed model is still required to be trained on a large number of previous tasks.

1.3 Memory-Augmented Networks

Memory-Augmented Neural Network (MANN, Santoro et al. (2016)) bridged the gap of leveraging an external memory for one-shot learning. MANN updates its external memory by learning a content-based memory writer. The Memory Module (Kaiser et al. (2017)) learns a Matching Network (Vinyals et al. (2016)) but includes an external memory that retains previously seen examples or their representatives (cluster of embeddings). Unlike MANN, the Memory Module has a deterministic way of updating its memory. Memory Module has the ability for providing ease and efficiency in grouping of incoming data and selecting class representations. However, Memory Module encounters limitation in learning and making precise predictions when the given memory space becomes extremely small. Our proposed STL focuses on the interaction between the per-task memory-augmented Fast Learners (FLs) and the Slow Predictor (SP) to optimize the data usage stored in the memory. This interaction allows an FL to learn better representations for a better lifelong and few-shot predictions.

1.4 Lifelong Learning

It is common for lifelong learning algorithms to store a form of knowledge from previously learned tasks to overcome forgetting. Some remember the task specific models (Lee et al. (2017)), while some store raw data, the hessian of the task, or the attention mask of the network for the task (Kirkpatrick et al. (2017); Lopez-Paz & Ranzato (2017); SerrÃ et al. (2018)). Some approaches such as Yoon et al. (2018) not only attempts to consolidate the model but also expands the network size. Other works like Hu et al. (2019); Schwarz et al. (2018) tried to solve the problem with fixed storage consumption. Except for Yoon et al. (2018), the previously mentioned works need to predefine the model capacity, and lacks the flexibility to unknown number of future tasks. Although Yoon et al. (2018) can expand its capacity when training for a new task, the challenge of deciding how many number of units to add during each expansion still remains.

Some of the recent models (Li & Hoiem (2016); Gepperth & Karaoguz (2016); Kirkpatrick et al. (2017); He & Jaeger (2018); Kemker & Kanan (2018); Sprechmann et al. (2018); Zenke et al. (2017)) in lifelong learning have taken inspiration on how the brain works (Parisi et al. (2019)). Our proposed framework is closely related to other *dual-memory systems* that are inspired by the *complementary learning systems* (CLS) theory, which defines the contribution of the hippocampus for quick learning and the neocortex for memory consolidation. A version of GeppNet (Gepperth & Karaoguz (2016)) that is augmented with external memory stores some of its training data for rehearsal after each new class is trained. FearNet (Kemker & Kanan (2018)) is composed of three networks for quick recall, memory consolidation, and network selection. Both GeppNet and FearNet have dedicated *sleep phases* for memory replay, a mechanism to mitigate catastrophic forgetting. STL, however, does not require a dedicated *sleep* or shutdown to consolidate the memory. This choice is based on considering that there are cases wherein a dedicated sleep time is not feasible, such as when using a machine learning model to provide a frontline service that needs to be up and running all the time and cannot be interrupted by a regular sleep schedule.

2 Technical Details

In this section, we discuss more technical details about the design and training of STL.

2.1 Solving Eq. (2)

There are different ways to solve Eq. (2). One can use either the gradient-based MAML (Finn et al. (2017)) or Reptile (Nichol & Schulman (2018)) to get an approximated solution efficiently. The constraint $\|\tilde{\theta} - \theta\| \leq R$ can be implemented by either adding a Lagrange multiplier in the objective or limiting the number of gradient steps in MAML/Reptile. In this paper, we use MAML due to its simplicity, ease of implementation, and efficiency, and

we enforce the constraint $\|\tilde{\theta} - \theta\| \leq R$ by limiting the number of adaptation steps of SP at runtime.

2.2 Empirical Loss of FLs

An FL in STL is compatible with different types of external memory modules, such as Santoro et al. (2016); Sprechmann et al. (2018); Vinyals et al. (2016). We choose the Memory Module (Kaiser et al. (2017)) in this paper due to its clustering capabilities, which increase space efficiency. For completeness, we briefly discuss how an FL based on the Memory Module are optimized.

Let $\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \cdots, \boldsymbol{h}^{(K)} \in \text{KNN}(g^{(s)}(x))$ be the sorted K nearest neighbors (from the closest to the farthest) of the embedding of x , and $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \cdots, \boldsymbol{v}^{(K)}$ be the corresponding values of the nearest neighbors in $\mathcal{M}^{(s)}$. We write $\mathcal{L}_{\text{FL}}(\phi^{(s)}; \mathcal{D}^{(s)}, \mathcal{M}^{(s)})$ as

$$\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}^{(s)}} [\langle g^{(s)}(\boldsymbol{x}; \phi^{(s)}), \boldsymbol{h}^{(b)} \rangle - \langle g^{(s)}(\boldsymbol{x}; \phi^{(s)}), \boldsymbol{h}^{(p)} \rangle + \varepsilon]_+,$$

where $\langle \cdot, \cdot \rangle$ denotes the cosine similarity between two vectors, and p and b are the smallest indices such that $\boldsymbol{v}^{(p)} = \boldsymbol{y}$ and $\boldsymbol{v}^{(b)} \neq \boldsymbol{y}$, respectively; $\boldsymbol{h}^{(p)}$ and $\boldsymbol{h}^{(b)}$ are the closest positive and negative neighbors. As this loss is minimized, $g^{(s)}$ maximizes the similarity of embedding of training data points to their positive neighbors, while minimizing the similarity to the negative neighbors by a margin of ε. The Memory Module Kaiser et al. (2017) also has deterministic update and replacement rules for records in $\mathcal{M}^{(s)}$. In effect, an $\boldsymbol{h}$ represents the embedding of *a cluster of* data points, and its value $\boldsymbol{v}$ denotes the shared label of points in that group.

2.3 Training

2.3.1 Alternate training

We sequentially train the STL for tasks $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \cdots$ coming in lifelong. For the current task $\mathcal{T}^{(s)}$, we train the STL using an alternate training approach. First, the weights $\phi^{(s)}$ of $g^{(s)}$ for $\mathcal{T}^{(t)}$ is updated by taking some descent steps following Eq. (3) in the main paper. Next, the θ that parametrizes f is updated following Eq. (2) in the main paper. One alternate training iteration involves training the FL for the current task for a steps, and then the SP for b steps. We set the alternate ratio $a : b$ to 1:1 by default. The pseudo-code of STL's training procedure is shown in Algorithms 1, 2, and 3.

2.3.2 Hyperparameter R

One important hyperparameter to decide before training the STL is R, which affects the number of adaptation steps used by SP. A larger R allows the adapted weights $\tilde{\theta}$'s to move farther from θ, which may lead the SP to better lifelong predictions but will result in higher computation cost at runtime. A smaller R helps stabilize θ after the model is trained on previous tasks and enables θ to guide the FLs for new incoming tasks sooner. We experimented on different values of R by adjusting the number of adaptation steps of SP, and found out that it does not need to be large to achieve good performance. Normally, it suffices to have less than 5 adaptation steps.

3 More Experiments

3.1 Space Efficiency

The SP in STL can work with FTs having very small external memory. Figure 8(a) shows the trade-off between the average all-task performance at the 10-th sequential task on the permuted MNIST dataset and the size (in number of embedded examples) of external memory. While the performance of most memory-augmented models drops when the memory size is 10, the SP can still perform well. This justifies that the invariant principles learned by the SP can indeed guide FLs to find better representations that effectively "bring back"

Algorithm 1 Training and prediction processes of STL

procedure STL_TRAIN($\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \cdots$)
  Initialize θ
  for every new incoming task $\mathcal{T}^{(s)}$ containing the training dataset $\mathcal{D}^{(s)}$ **do**
    Initialize $(\phi^{(s)}, \mathcal{M}^{(s)})$
    repeat ▷ Alternate training process
      Update $(\phi^{(s)}, \mathcal{M}^{(s)})$ using FL_Train($(\phi^{(s)}, \mathcal{M}^{(s)})$; $\mathcal{D}^{(s)}$, θ)
      Update θ using SP_Train(θ; $\{(\phi^{(t)}, \mathcal{M}^{(t)})\}_{t=1}^{s}$)
    until convergence or maximum iteration is reached
  end for
end procedure
procedure STL_PREDICT($\boldsymbol{x}, t$) ▷ t is the task ID
  Embed the input: $\boldsymbol{h} \leftarrow g^{(t)}(\boldsymbol{x})$
  $\hat{\boldsymbol{y}}_{\text{SP}} \leftarrow$ SP_Predict($\boldsymbol{h}$; $\mathcal{M}^{(t)}$, θ)
  $\hat{\boldsymbol{y}}_{\text{FL}} \leftarrow$ FL_Predict($\boldsymbol{h}$; $\mathcal{M}^{(t)}$)
  Return $\hat{\boldsymbol{y}}_{\text{SP}}$, $\hat{\boldsymbol{y}}_{\text{FL}}$
end procedure

Algorithm 2 Training algorithms for FLs and SP

procedure FL_TRAIN($(\phi^{(s)}, \mathcal{M}^{(s)})$; $\mathcal{D}^{(s)}$, θ)
  Sample a batch $(\mathcal{X}, \mathcal{Y})$ from $\mathcal{D}^{(s)}$
  Get embeddings of $\mathcal{X}$ from FL: $\mathcal{H} \leftarrow g^{(s)}(\mathcal{X}; \phi^{(s)})$
  Get FL predictions: $\hat{\mathcal{Y}}_{\text{FL}} \leftarrow$ FL_Predict($\mathcal{H}; \mathcal{M}^{(s)}$)
  Get SP predictions: $\hat{\mathcal{Y}}_{\text{SP}} \leftarrow$ SP_Predict($\mathcal{H}; \mathcal{M}^{(s)}, \theta$)
  Calculate total loss: $\mathcal{L} \leftarrow \text{loss}(\hat{\mathcal{Y}}_{\text{FL}}, \mathcal{Y}) + \lambda\, \text{loss}(\hat{\mathcal{Y}}_{\text{SP}}, \mathcal{Y})$
  Update parameter of FL: $\phi^{(s)} \leftarrow \phi^{(s)} - \alpha \nabla_{\phi^{(s)}} \mathcal{L}$
  Update memory $\mathcal{M}^{(s)}$ by adding key-value pairs $(\mathcal{H}, \hat{\mathcal{Y}}_{\text{FL}})$
end procedure
procedure SP_TRAIN(θ; $\{(\phi^{(t)}, \mathcal{M}^{(t)})\}_{t=1}^{s}$)
  for all task $\mathcal{T}^{(t)}$, $t \leq s$, trained so far **do**
    Sample a meta-batch $(\mathcal{X}^{(t)}, \mathcal{Y}^{(t)})$ from $\mathcal{M}^{(t)}$
    Get SP predictions: $\hat{\mathcal{Y}}_{\text{SP}}^{(t)} \leftarrow$ SP_Predict($\mathcal{X}^{(t)}; \mathcal{M}^{(t)}, \theta$)
  end for
  Update parameter of SP: $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{t \leq s} \text{loss}(\hat{\mathcal{Y}}_{\text{SP}}^{(t)}, \mathcal{Y}^{(t)})$
end procedure

Algorithm 3 Prediction procedure of FLs and SP

procedure FL_PREDICT($\boldsymbol{h}$; $\mathcal{M}^{(t)}$)
  Retrieve top-K nearest neighbors, KNN($\boldsymbol{h}$), and their corresponding values, $\mathcal{V}$, from $\mathcal{M}^{(t)}$
  Get FL prediction: $\hat{\boldsymbol{y}}_{\text{FL}} \leftarrow$ weighted average of the values $\mathcal{V}$
  Return $\hat{\boldsymbol{y}}_{\text{FL}}$
end procedure
procedure SP_PREDICT($\boldsymbol{h}$; $\mathcal{M}^{(t)}$, θ)
  Retrieve top-K nearest neighbors, KNN($\boldsymbol{h}$), and their corresponding values, $\mathcal{V}$, from $\mathcal{M}^{(s)}$
  Initial parameter to be fine-tuned: $\tilde{\theta} \leftarrow \theta$
  repeat ▷ Runtime adaptation
    $\hat{\mathcal{Y}} \leftarrow f(\text{KNN}(\boldsymbol{h}); \tilde{\theta})$
    $\tilde{\theta} \leftarrow \tilde{\theta} - \alpha \nabla_{\theta} \text{loss}(\hat{\mathcal{Y}}, \mathcal{V})$
  until maximum fine-tuning step (based on R) is reached
  Get SP prediction after fine-tuning: $\hat{\boldsymbol{y}}_{\text{SP}} \leftarrow f(\boldsymbol{h}; \tilde{\theta})$
  Return $\hat{\boldsymbol{y}}_{\text{SP}}$
end procedure

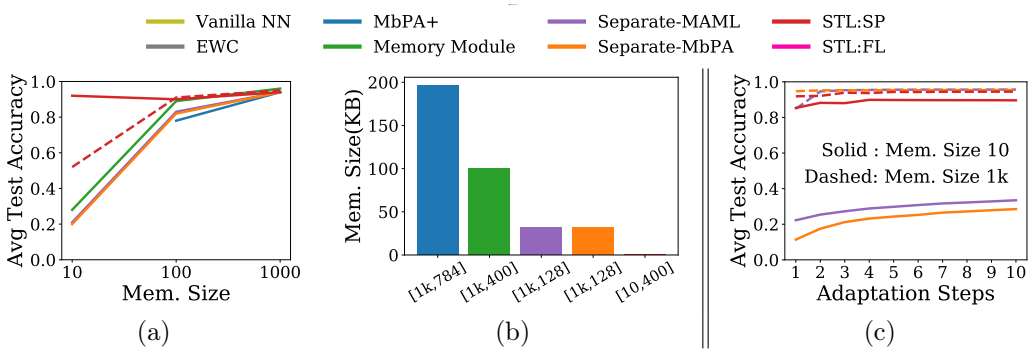


Figure 8: (a) Trade-off between the average all-task performance at the 10-th sequential task on the permuted MNIST dataset and the size (in number of embedded examples) of external memory. (b) Space consumption in order to achieve at least 0.9 average all-task accuracy. A pair $[a, b]$ denotes a embedded examples, each with b features, in memory. (c) Performance gain of different adaptive models after runtime adaptation.

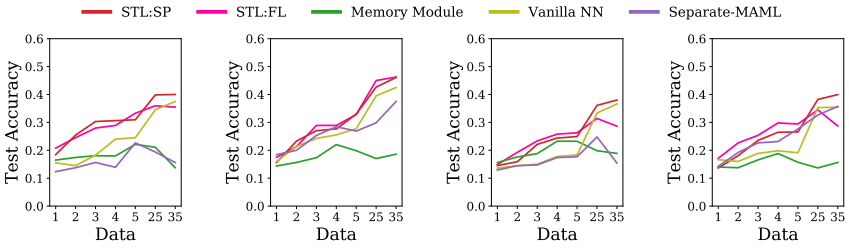


Figure 9: Performance for the (a) 2-nd, (b) 3-rd, (c) 4-th, and (d) 5-th sequential CIFAR-100 Hard task that has only a few shots (in number of batches) of training data.

the knowledge of SP for a particular task. Figure 8(b) shows the memory space required by different models in order to achieve at least 0.9 average all-task accuracy. The STL consumes less than 1% space as compared to MbPA and Memory Module.

3.2 Adaptation Efficiency

The SP also has high adaptation efficiency. Figure 8(c) shows the performance gain of different adaptive models after runtime adaptation. When the memory size is 1000, all models can adapt for the current task to give improved performance. However, the adaptation efficiency of the baseline models drops when the memory size is 10. The SP, on the other hand, achieves good performance in this case even after being adapted for just *one* step thanks to 1) the SP is trained to be ready for adaptation and 2) the invariant principles learned by the SP are useful by themselves and require only few examples (embeddings) to transform to a good hypothesis.

3.3 Sequential Few-shot Learning on CIFAR Hard

Figure 9 shows the sequential few-shot learning results of different models on the CIFAR Hard dataset. In this dataset, the labels of different tasks come from different superclasses in the original CIFAR 100 dataset. So, it is very challenging for a model to learn from only a few examples in a new task without being able to see a lot of previous tasks. As we can see, our STL model still outperforms other baselines. In particular, Figure 9(a) shows that the STL is able to perform few-shot learning after seeing just one previous task. Again, the above demonstrates that the interactions between the FLs and SP are a key to improving machine ability for the joint lifelong and few-shot learning.

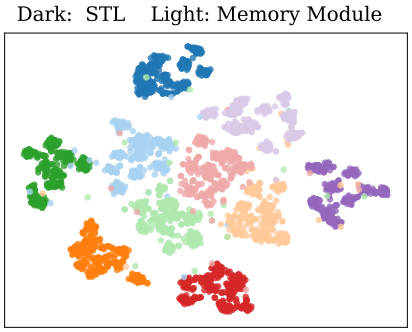


Figure 10: t-SNE visualization of the embeddings of examples saved in different FLs for the 5 CIFAR-100 Normal tasks. Different colors represent different tasks.

3.4 Visualization

In order to understand how the feedback from SP guide the representation learning of FLs, we visualize the embeddings stored in the FLs and the Memory Module. We sample 300 testing examples per task, get their embeddings from the two models, and then project them onto a 2D space using the t-SNE algorithm (Maaten & Hinton (2008)). The results are shown in Figure 10. We can see that the embeddings produced by different FLs for different tasks are more distant from each other than those output by the Memory Module. Recall that the feedback term in Eq. (3) encourages each FL to learn features that help the SP adapt. Therefore, each FL learns more salient features for the respective task so the SP will not be confused by two tasks having similar embeddings. This, in turn, quickly stabilizes SP and makes it an efficient guide (through the feedback term) to learning the FL for a new task when there are only few examples and also few previous tasks, as Figure 3 shows.