Continual Sequence Generation with Adaptive Compositional Modules

Yanzhe Zhang
Georgia Institute of Technology
z_yanzhe@gatech.edu

Xuezhi Wang
Google
xuezhiw@google.com

Diyi Yang
Georgia Institute of Technology
dyang888@gatech.edu

Abstract

Continual learning is essential for real-world deployment when there is a need to quickly adapt the model to new tasks without forgetting knowledge of old tasks. Existing work on continual sequence generation either always reuses existing parameters to learn new tasks, which is vulnerable to catastrophic forgetting on dissimilar tasks, or blindly adds new parameters for every new task, which could prevent knowledge sharing between similar tasks. To get the best of both worlds, in this work, we propose continual sequence generation with adaptive compositional modules to adaptively add modules in transformer architectures and compose both old and new modules for new tasks. We also incorporate pseudo experience replay to facilitate knowledge transfer in those shared modules. Experiment results on various sequences of generation tasks show that our framework can adaptively add modules or reuse modules based on task similarity, outperforming state-of-the-art baselines in terms of both performance and parameter efficiency. We make our code public at https://github.com/GT-SALT/Adaptive-Compositional-Modules.

1 Introduction

Current state-of-the-art language generation models can achieve great performance on a wide range of sequence generation tasks (Radford et al., 2019; Lewis et al., 2020) with a static data distribution. However, real-world scenarios are often changing which requires the model to learn with dynamic data distributions. In such cases of data distributions shift, current generation models often suffer from *catastrophic forgetting* (Sun et al., 2019): models completely and abruptly forget previously learned information upon learning new information. Continual learning (CL) (Ring, 1998; Thrun, 1998) has been introduced to improve model's ability to learn tasks in a stream by mitigating forgetting
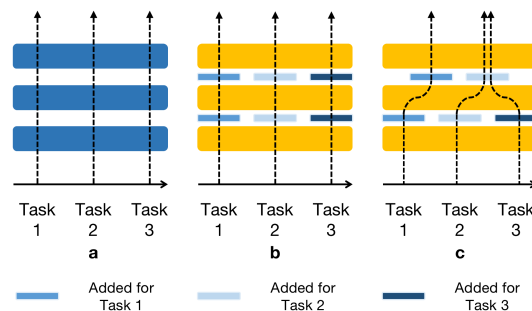

Figure 1: Comparison between previous methods (a and b) and our proposed method (c), from a multi-layer transformer model perspective. The blue blocks refer to learnable modules and the yellow blocks refer to frozen pretrained modules . **a**: retrain the whole model every time when new tasks arrive. **b**: insert task-specific modules for each task, while keeping the pretrained model frozen. **c**: detect reusable old modules and add new modules adaptively.

and facilitating knowledge transfer (Lopez-Paz and Ranzato, 2017), however, continual sequence generation is relatively under-investigated.

Comparing to continual learning on text classification and question answering (Wang et al., 2020; Holla et al., 2020; Huang et al., 2021), continual sequence generation is more challenging, since the output is no longer discrete labels but sequential text data in different styles/domains. Based on how to retain old knowledge while learning new tasks, current continual sequence generation methods can be categorized into two types. The first one continually learns new tasks on old parameters (Fig 1 a), with approaches like experience replay (Sun et al., 2019; Chuang et al., 2020) and regularization (Mi et al., 2020) to maintain old knowledge. However, since all tasks share the same parameters, some degree of interference between tasks is unavoidable. Another line of work continually inserts new task-specific modules (adapters proposed by Houlsby et al., 2019) into every transformer layer for every new task while freezing pretrained mod-

els and modules used by old tasks (Fig 1 b, Madotto et al., 2021), which might prevent knowledge transfer between tasks and introduce possible parameter redundancy. In this work, we aim to get the best of both worlds: how to encourage the models to reuse modules from previous tasks as much as possible and to only add new modules if needed?

To this end, we propose continual sequence generation with ***adaptive compositional modules***, as shown in Fig 1 c. Specifically, we introduce a two-stage process for every new coming task: a decision stage and a training stage. During decision stage, we decide which modules to reuse and whether we need to add a new module. During training stage, the model architecture is determined and fixed. We augment new task's training process with pseudo experience replay (Sun et al., 2019) to further mitigate forgetting and facilitate knowledge transfer in those shared layers. Our model architecture is *adaptive*, as it can automatically add new modules for dissimilar tasks and reuse modules for similar tasks, thus making it robust to different scenarios of continual learning. Furthermore, it is *compositional* because for every new task, our new architecture is composed of reused modules from old tasks and newly added modules, which allows knowledge reuse and transfer.

To evaluate the above adaptive compositional framework, we experiment with four representative sequence generation tasks following prior work (Sun et al., 2019; Chuang et al., 2020): natural language generation, SQL query generation, summarization and task-oriented dialogue arriving in a stream. Different from prior work that only tests their methods on very short task sequences or long task sequences with similar tasks only, we validate our approach on longer sequences containing diverse tasks with different levels of similarity. We believe this is a suitable scenario to validate both the model's ability to mitigate forgetting and its ability to facilitate knowledge transfer. In summary, this work makes two key contributions: (1) We propose continual sequence generation with adaptive compositional modules, to maximize knowledge transfer via module-reusing while adaptively adding new modules to mitigate task-interference and catastrophic forgetting. (2) Experiments with longer and more task sequences show that our approach outperformed baselines with higher parameter efficiency.

2 Related Work

Continual Learning Without allocating new parameters for new tasks, prior work mainly leverages experience replay (Wang et al., 2019; Sun et al., 2019) and regularization to mitigate catastrophic forgetting. In experience replay, models are retrained on old examples from previous tasks while learning new tasks. Those old examples are usually stored in a fixed size (Mi et al., 2020) or expanding (Huang et al., 2021) memory buffer. Besides replaying old examples, regularization on the hidden states (Wang et al., 2019; Han et al., 2020; Huang et al., 2021) or parameters (Mi et al., 2020) could be further added to prevent severe distortion. Another line of work is to create new parameters for new tasks while freezing parameters used by old tasks. In computer vision, progressive neural network (Rusu et al., 2016) continually adds new branches of parameters for new image classification tasks with lateral connections to facilitate forward knowledge transfer. Dynamically expandable network (Yoon et al., 2017) expands neural networks at neuron level by using regularization to restrict the number of added neurons. While allocating a big network in advance, PackNet (Mallya and Lazebnik, 2018) continually assigns a parameter subset to each task by network pruning.Li et al. (2019) employ neural architecture search (Liu et al., 2018) to optimize on new task's structure before learning new tasks. In language domain, prior work often utilizes adapter (Houlsby et al., 2019; Madotto et al., 2021; Ermis et al., 2022), which could be considered as task-specific MLPs inserted into frozen transformer layers. However, since all adapter modules are designed for only one specific task, no knowledge transfer is directly allowed in this case. Extra modules like attention module (Pfeiffer et al., 2021), capsule network (Ke et al., 2021), and hypernetworks (Jin et al., 2021) are demonstrated beneficial for knowledge transfer, but they need to introduce extra parameters and fail to consider any reusable or compositional modules.

Avoiding privacy concerns, this work also follows a line of work that doesn't store real examples for experience replay, such as generating examples by GAN (Atkinson et al., 2018), synthesizing examples (Xu et al., 2022) by model-inversion (Smith et al., 2021b), and using unlabeled data in the learning environment (Smith et al., 2021a). In language domain, LAMOL (Sun et al., 2019) trains the language model to solve current tasks and generate

current training examples simultaneously, then this model can generate "pseudo" old examples for replay before any new tasks. We adopt this pseudo experience replay along to alleviate the forgetting in the shared modules of our approach.

Continual Learning for Sequence Generation Building on an auto-regressive language model, LAMOL (Sun et al., 2019) makes initial exploration on continual sequence generation. On the basis of LAMOL, knowledge distillation (Chuang et al., 2020; Sun et al., 2020) is shown to be effective via improving knowledge transfer while changing tasks. ARPER (Mi et al., 2020) combines regularization on parameters (Kirkpatrick et al., 2017) with prioritized exemplar replay. Keeping the pretrained model frozen, Madotto et al. (2021) added task-specific modules for each task together with a perplexity-based classifier, without taking into account the potential for knowledge transfer between different tasks. Instead of blindly adding new modules for new tasks, our approach can detect reusable modules and strategically add new adapter modules in those layers in which reusing old modules would lead to severe forgetting. Without introducing extra knowledge transfer modules, our approach enables knowledge transfer via module sharing.

Task-specific Modules Traditional finetuning approaches (Peters et al., 2018; Devlin et al., 2019; Radford et al., 2019) usually modify all the parameters in large pretrained modules while learning downstream tasks. Recently, a line of work has been proposed to improve the parameter-efficiency of finetuning by inserting task-specific modules into freezing pretrained models. Adapter (Houlsby et al., 2019) inserts MLP layers into each transformer layer. PrefixTuning (Li and Liang, 2021) prepends key-value pairs to each transformer layer as activations. Prior work also shows that these task-specific modules might benefit from a more adaptive usage. For example, AdapterDrop (Rücklé et al., 2021) shows that removing adapters from lower transformer layers can almost maintain the original performance while reducing computational overhead. Guo et al. (2021) leveraged latent variables to decide whether to skip adapter modules in certain transformer layers to speed up decoding. However, our approach goes beyond the notion of "task-specific", recomposes reusable modules from different tasks, and learns compositional architectures for new coming tasks.

3 Background

Continual Generation Formulation Assuming multiple sequence generation tasks $\{T_1...T_n\}$ arrive in a stream, each task T_i has a set of training examples $\{P_1^i, P_2^i..., P_k^i\}$, where P_j^i denotes a $(input, output)$ pair in Task i. While learning on task T_i $(i > 2)$, we have **no access** to examples from previous tasks. The final goal is to optimize the model's average performance on **all** tasks after training on the whole sequence.

Finetuning In order to integrate different sequence generation tasks into a single framework, we use **finetuning** as a general strategy. On the basis of an autoregressive language model, the core idea is to feed the model $input$ and train the model to generate the corresponding $output$ subsequently. To distinguish between tasks, we add an extra $question$ following every $input$ to describe the purpose of each task. For example, the $question$ for natural language generation tasks is *What is the natural language form?* Formally, for each $(input, question, output)$ triple, the model is optimized to generate the corresponding $output$ given $input$ and $question$:

$$L_{finetune}(x) = \sum_{t=m+1}^{n} -\log P(x_t|x_{<t})$$

where $x = \{x_1, ..., x_n\}$ denotes the concatenation of $input$, $question$ and $output$, and $\{x_1, ..., x_m\}$ refers to $input$ and $question$.

Adapter The *module* used in our framework refers to adapter (Houlsby et al., 2019), which is a task-specific module inserted into each frozen pretrained transformer layers (Vaswani et al., 2017). In addition to residual connection (He et al., 2016) and layer normalization (Ba et al., 2016), one transformer layer contains two primary sub-layers: an attention layer and a feed forward layer. One adapter module consists of two multi-layer perceptrons (MLP), one (MLP_{MH}) following the multi-head attention layer and one (MLP_{FF}) following the feed forward layer.

4 Two-Stage Methods

Motivated by prior continual sequence generation work (Madotto et al., 2021) that uses Adapter (Houlsby et al., 2019) to insert new adapter module

into every transformer layer for each new coming task, we propose to strategically *decide* whether we can reuse some adapter modules from old tasks before *training* on each new coming task, in a two-stage manner: decision stage and training stage, where the former determines the architecture for new tasks and the later trains the model.

4.1 Decision Stage

The decision stage aims to answer two questions: do we need to add a new module in this layer? If not, which old modules should we reuse? Inspired by interpolation-based data augmentation (Chen et al., 2020, 2021) and neural architecture search (Liu et al., 2018), we utilize **Hidden State Mixing** for module selection. Assume that there are several modules as potential candidates to be selected, after calculating their output separately, we calculate their *weighted average* as the overall output, which is then passed to the next part of the model (See the left part in Figure 2). After training the entire model end-to-end, we assume that the module with the largest learned weight is the most useful one, and thus will be selected for the reuse.

Formally, assume that we already have inserted k modules into the lth transformer layer, each consisting of two MLPs: $(MLP_{MH}^{1,l}, MLP_{FF}^{1,l})$... $(MLP_{MH}^{k,l}, MLP_{FF}^{k,l})$. At the beginning of decision stage, we add one more module $(MLP_{MH}^{k+1,l}, MLP_{FF}^{k+1,l})$. Given these learnable weight coefficients $[\lambda_{1,l}, \ldots, \lambda_{k+1,l}]$, multi-head attention layer output o_{mh}^l, the feed forward layer output o_{ff}^l, we mix the hidden states as follow:

$$h_{mh}^l = \sum_{t=1}^{k+1} \lambda_{t,l} MLP_{MH}^{t,l}(o_{mh}^l)$$

$$h_{ff}^l = \sum_{t=1}^{k+1} \lambda_{t,l} MLP_{FF}^{t,l}(o_{ff}^l)$$

where both h_{mh}^l and h_{ff}^l are then fed into their following *Add & Norm* layers. To ensure $\sum_{t=1}^{k+1} \lambda_{t,l} = 1$, we use *softmax* function to produce $\lambda_{1,l}, \ldots, \lambda_{k+1,l}$ from $c_{1,l}, \ldots, c_{k+1,l}$:

$$\lambda_{i,l} = \frac{e^{c_{i,l}}}{\sum_{t=1}^{k+1} e^{c_{t,l}}}, i = 1 \ldots k+1$$

Using this mixing approach in every transformer layer, we optimize our model using L_{train} (see Sec 4.2) for the new task and find the most suitable modules for each layer. Note that (i) In this process, the pretrained model and all old modules are frozen, and only mixing coefficients and newly added modules will be learned. (ii) Calculating the weighted average is a convenient approximation of using one adapter at a time, which is the real setting during training stage and inference. (iii) Comparing to other baselines in Figure 1, introduced decision stage to decide the architecture does introduce extra computation, while computation of different MLPs at one position is parallelizable to speed up.

To avoid the learned weight coefficient $\lambda_{1,l}, \ldots, \lambda_{k+1,l}$ to be too close to a uniform distribution in certain layers, we further add an additional regularization term to L_{train}, which is the sum of *entropy* of every discrete probability distribution $[\lambda_{1,l}, \ldots, \lambda_{k+1,l}]$:

$$L_{entropy} = \gamma \sum_l \sum_{i=1}^{k+1} -\lambda_{i,l} \log(\lambda_{i,l})$$

where γ is a coefficient tuned as a hyper-parameter.

In this stage, a trivial solution could be allocating a new module in every layer regardless of whether old modules are reusable. To avoid this trivial solution and reuse shareable modules as much as possible, we design a prior using the ***initialization of the coefficient weights***. For every l, $c_{1,l}...c_{k,l}$ is initialized to c ($c > 0$), while $c_{k+1,l}$ is initialized to $-c$. After *softmax*, the weight of each old module is e^{2c} times the weight of the new module, increasing the tendency to reuse old modules.

4.2 Training Stage

We further incorporate pseudo experience replay (Sun et al., 2019) to mitigate forgetting and facilitate knowledge transfer in those shared modules. The main idea is to teach a generative model to solve current task and to generate current task's examples **simultaneously**. Then before training on each new task, we can generate a set of pseudo old examples and replay them during training.

Thus, in addition to the finetuning loss to solve each task, we introduce an extra loss L_{gen} for the model to generate current task's examples. Formally, given the whole sequence of $x = \{input, question, output\}$, we first add a special token [GEN] at the beginning of x to form a new sequence x', and then optimize the model as follows:

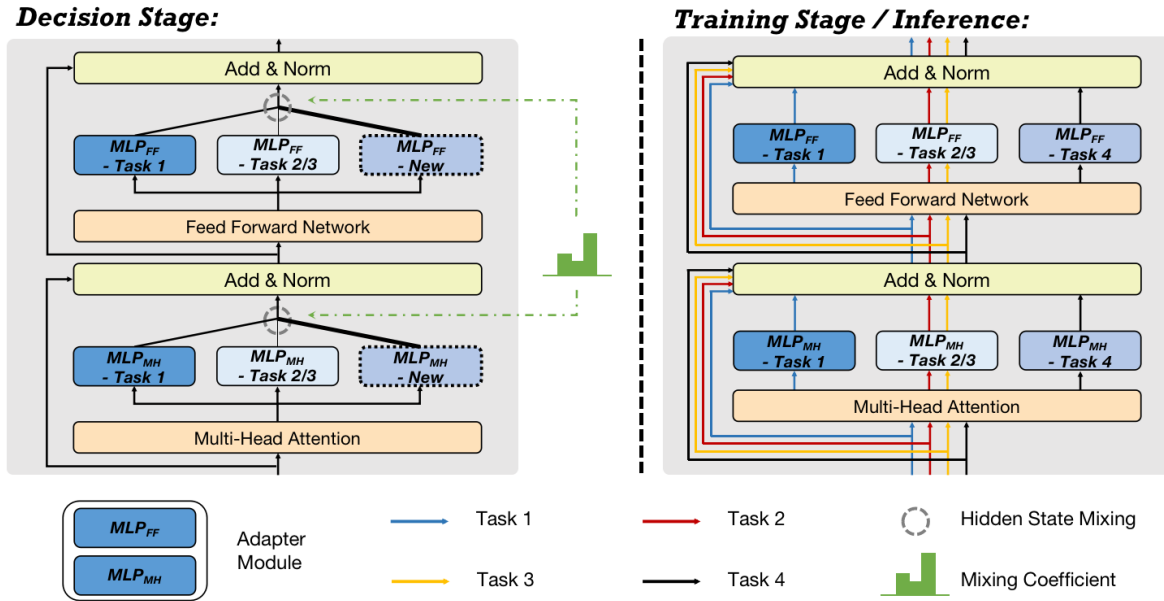$$L_{gen}(x') = \sum_{t=1}^{n+1} -\log P(x_t'|x_{<t}')$$

Figure 2: Our proposed model architecture with adaptive compositional modules for transformer layers. Assume after learning three tasks (1, 2, 3), we have one module for task 1, and another for task 2 and 3 in this layer. **Left**: During decision stage for task 4, we first insert a new module at this position, then all inserted modules will be used for selection using hidden state mixing. **Right**: Assume that we finally decide to add one module at this position, then each task would use its own architecture during training stage and inference.

Note that we use different special tokens for different tasks, thus we can generate examples for specified tasks afterwards. Combining with the finetune loss, the overall training loss is:

$$L_{train} = L_{finetune} + \eta L_{gen}$$

where η is the weight for the L_{gen} loss.

Once our model has the ability to generate "pseudo" examples from old tasks, another question is *When to generate "pseudo" examples?* Since those "pseudo" examples are for shared modules between old tasks and the current task, we only generate them while some old modules are reused for the current task. In that case, we train our model using L_{train} on the current dataset together with the generated examples. Otherwise, there is no need for pseudo experience replay and we just train our model using L_{train} on the current dataset.

5   Experiments

5.1   Datasets

Following Sun et al. (2019) and Chuang et al. (2020), we evaluate our approach on four representative sequence generation tasks: natural language generation, SQL query generation, summarization and task-oriented dialogue modeling. Specifically,

we test our proposed approach under two common scenarios: (1) *CL on similar tasks*: in this case, the new coming tasks often share the same task pattern with learned tasks, but are from different domains. We use E2ENLG (Novikova et al., 2017) and four different domains (restaurant, hotel, tv, laptop) from RNNLG (Wen et al., 2015) to form five *similar* tasks. Then we use four different orders of these tasks as our testing task sequences. (2) *CL on dissimilar tasks*: in this case, the distribution shift between new tasks and old tasks could be relatively large, so the major challenge is to retain old knowledge as much as possible while learning new tasks. In this case, we further incorporate WikiSQL (SQL query generation, Zhong et al., 2017), CNN/DailyMail (news article summarization See et al., 2017), MultiWOZ (semantic state sequence generation (Budzianowski et al., 2018)) into our task sequences[1]. We randomly pick four different orders as our testing task sequences. In total, we use eight different task sequences (Table 1) to evaluate our models. The statistics/metrics for each dataset and the finetuing results are in Appendix A.

---

[1]We use "e2e" for E2ENLG, "rest" for RNNLG (restaurant), "hotel" for RNNLG (hotel), "tv" for RNNLG (tv), "laptop" for RNNLG (laptop), "wiki" for WikiSQL, "cnn" for CNN/DailyMail, "woz" for MultiWOZ.

Order	Task Sequence
1	e2e → rest → hotel → tv → laptop
2	laptop → tv → hotel → rest → e2e
3	rest → tv → e2e → laptop → hotel
4	hotel → e2e → rest → laptop → tv
5	woz → cnn → e2e → rest → hotel
6	e2e → wiki → hotel → woz → rest
7	hotel → e2e → woz → wiki → cnn
8	cnn → hotel → wiki → e2e → woz

Table 1: Eight random different task sequences. The first 4 includes different orders of similar tasks, the last 4 includes different orders including dissimilar tasks.

5.2 Baselines

We compare our proposed model with the following baselines: (i) **Finetune** (Yogatama et al., 2019): We finetuned GPT-2 model on several tasks sequentially. (ii) **EWC** (Kirkpatrick et al., 2017) added regularization on parameters according to their importance to old tasks. (iii) **LAMOL** (Sun et al., 2019) finetuned the whole GPT-2 model continually with the help of pseudo experience replay. (iv) **Adapter+CL** (Madotto et al., 2021) inserted adapter (Houlsby et al., 2019) modules into every GPT-2's layer for each task. (v) **Adapter+Drop** (Rücklé et al., 2021): We removed all those adapter modules from the first three layers in GPT-2 based on Adapter+CL. (vi) **Adapter+LAMOL**: We only inserted adapter modules into every transformer layer for the first task, then used those adapter modules to learn the whole the task sequence with pseudo experience replay. Note that ARPER (Mi et al., 2020) also tackles the problem of continual sequence generation, but it needs an extra memory buffer to store examples from old tasks, which is not comparable with ours.

Implementation Details We use GPT-2 (Radford et al., 2019) in HugginceFace Transformers (Wolf et al., 2020) as our backbone and adapter implementation by AdapterHub (Pfeiffer et al., 2020). More details can be found in Appendix A.

6 Results and Analysis

To evaluate the overall performance on all tasks, we use the mean of all tasks' performance score following Sun et al. (2019); Mi et al. (2020); Madotto et al. (2021). For each scenario (*similar* tasks and *dissimilar* tasks), we report the average of mean scores on all sequences as an overall metric. Be-

yond these, we also provide (i) evaluation results using *geometric mean* and (ii) final performance of each task in Appendix A. Table 2 summarizes the final performance on all eight task sequences. We observed that finetuning sequentially suffered from very severe forgetting, no matter on *similar* or *dissimilar* tasks, highlighting the importance of continual learning work. Though EWC can significantly increase the performance of finetuning, its performance is still far behind LAMOL, highlighting the importance of experience replay.

For sequences containing *similar* tasks, the performance of Adapter+CL is inferior to Adapter+LAMOL even with more learnable parameters. This indicates that sharing parameters and experience replay can further facilitate knowledge transfer when tasks are similar. On the premise of pseudo experience replay, our method performs better than Adapter+LAMOL, demonstrating the effectiveness of our adaptive compositional architecture. Our approach also achieves a much higher parameter efficiency than Adapter+CL and Adapter+Drop. For sequences containing *dissimilar* tasks where the transferable knowledge is limited and parameter sharing might cause degradation, Adapter+CL and Adapter+Drop seem more robust compared to Adapter+LAMOL and LAMOL, since they avoid catastrophic forgetting by parameter isolation. Using a similar number of parameters to Adapter+Drop, our method outperforms Adapter+CL consistently on all task sequences, confirming that our method can prevent interference between dissimilar tasks while reducing parameter redundancy.

6.1 Ablation Studies

We randomly selected task sequence #1 from *similar* tasks and sequence #8 from sequences of *dissimilar* tasks for our ablation studies.

Importance of Each Component To examine the importance of each component in our method, we experiment with different settings: not using entropy loss (w/o Entropy Loss), initializing all weight coefficients with zero (w/o Weight Ini), and not replaying pseudo data (w/o Pseudo ER). As shown in Table 3, we found that (i) After removing entropy loss, the performance on sequence #1 is almost maintained by using more parameters. Meanwhile, the performance on sequence #8 drops significantly while using the same number of parameters. This observation suggests that the en-

Methods		Finetune	EWC	LAMOL	Adapter +CL	Adapter +Drop	Adapter +LAMOL	Ours
Pseudo Experience Replay		✗	✗	✓	✗	✗	✓	✓
Similar Tasks	# 1	43.0	56.9	66.3	64.2	63.9	65.9	**66.1**
	# 2	37.0	47.9	67.0	64.2	63.9	66.2	**66.5**
	# 3	51.7	61.4	66.6	64.2	63.9	65.6	**65.8**
	# 4	45.0	58.3	66.6	64.2	63.9	65.2	**65.7**
Avg Performance		44.2	56.2	66.6	64.2	63.9	65.7	**66.0**
Avg Learnable Para.		124.45M	124.45M	124.45M	8.95M	6.71M	1.79M	2.44M
Dissimilar Tasks	# 5	33.6	37.5	57.0	57.5	57.4	54.3	**58.2**
	# 6	32.6	37.9	62.5	64.9	64.5	62.2	**65.9**
	# 7	19.7	37.5	56.7	57.3	56.7	54.6	**58.3**
	# 8	26.3	38.8	56.8	57.3	56.7	53.8	**58.2**
Avg Performance		28.1	37.9	58.3	59.3	58.8	56.2	**60.1**
Avg Learnable Para.		124.45M	124.45M	124.45M	8.95M	6.71M	1.79M	6.60M

Table 2: The mean of final performance score on all tasks. We use two random seeds for each task sequence. Note that the final performance of Adapter+CL and Adapter+Drop is not affected by task ordering within the same group of tasks. For each sequence, we mark the best representation in **bold**, where LAMOL is not compared due to the difference in the order of magnitude of the learnable parameters. For each scenario, the p-values of paired t-test between 8 numbers of our approach and the second highest comparable baseline is smaller than 0.05, demonstrating significant improvement.

Method	Sequence #1 Avg	Sequence #1 Avg L.P.	Sequence #8 Avg	Sequence #8 Avg L.P.
Ours	66.1	2.24M	58.2	6.49M
w/o Entropy loss	66.1	2.54M	57.6	6.49M
w/o Weight Ini	64.2	7.09M	57.7	8.65M
w/o Pseudo ER	43.2	2.08M	55.9	6.34M

Table 3: Ablation study on (i) entropy loss (ii) weight initialization (iii) pseudo experience replay. The left part includes results for sequence #1 while the right part includes result for sequence #8. Note that "Avg" refers to the mean of performance score on all tasks and "Avg L.P." refers to the mean of learnable parameters.

Length	Adapter +CL	Adapter +LAMOL	Ours
2 Tasks(#1)	56.8 (+0.0)	57.5 (+0.8)	57.7 (+0.9)
3 Tasks(#1)	59.5 (+0.0)	60.3 (+0.6)	60.1 (+0.5)
4 Tasks(#1)	62.3 (+0.0)	63.5 (+1.3)	63.7 (+1.6)
5 Tasks(#1)	64.2 (+0.0)	65.9 (+2.0)	66.1 (+2.1)
2 Tasks(#8)	45.4 (+0.0)	46.2 (+1.3)	46.0 (+1.2)
3 Tasks(#8)	51.3 (+0.0)	51.9 (+0.8)	52.3 (+0.9)
4 Tasks(#8)	50.9 (+0.0)	49.7 (-1.7)	51.8 (+0.6)
5 Tasks(#8)	57.3 (+0.0)	53.8 (-4.6)	58.2 (+0.5)

Table 4: Impact of the task sequence length. Note that "n Tasks(#i)" refers to after sequentially training on the first n tasks in sequence #i, we report the mean of performance score on those n tasks and the backward transfer in parentheses.

tropy loss is beneficial to achieve a better trade-off between adding parameters and maintaining good performance. (ii) When we initialize all weight co-efficients with zero, there is no explicit tendency to reuse old examples. In this case, many redundant modules are created thus preventing knowledge transfer, which leads to performance drop on both sequences. The drop on sequence #1 is more se-vere due to there is more transferable knowledge between similar tasks. We therefore conclude that weight initialization is important to enable knowl-edge transfer between similar tasks. (iii) Removing pseudo experience replay leads to the most severe performance drop on both sequences. Though our approach strategically detect which modules can be reused, directly training them on new tasks without protecting old knowledge will lead to catastrophic

forgetting.

Impact of Task Sequence Length Prior work in continual learning (Madotto et al., 2021; Huang et al., 2021) suggests that the differences in se-quence length could influence the performance of continual learning. To this end, we further inves-tigated the impact of sequence length in Table 4, where we reported the average performance at ev-ery step and calculated *Backward Transfer* follow-ing Lopez-Paz and Ranzato (2017):

$$BWT_k = \frac{1}{k-1}\mathbb{E}_{i=1...k-1}(R_{k,i} - R_{i,i})$$

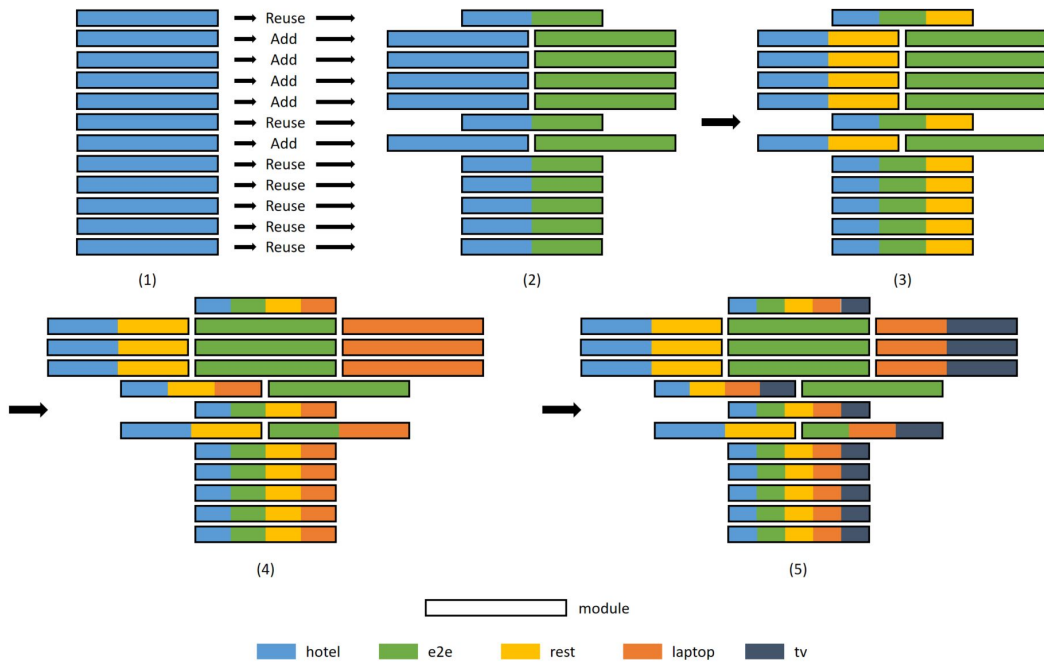where $R_{i,j}$ is the performance score on the jth task after training on the ith task.

Figure 3: The growing process of our model on sequence: hotel → e2e → rest → laptop → tv. The 1st layer is shown at the bottom and the 12th layer is at the top of each figure. Note that here we only depict the architecture growing process of our inserted modules: (i) Each rectangle represents a module *added* in that specific transformer layer. (ii) Each module is painted with the corresponding color if it is used by a task. (iii) Modules with multiple colors are *shared* by multiple tasks.

We found that, on sequence #1, Adapter+LAMOL and our method consistently outperform Adapter+CL in all stages, which could be explained by better knowledge transfer between multiple tasks. Beyond that, our method outperforms Adapter+LAMOL in most cases, demonstrating the benefits of adaptively adding modules. On sequence #8, Adapter+LAMOL struggles when the length of task sequence becomes longer. As more and more tasks arrive, the impact of task dissimilarity and distribution shift gets larger that pseudo experience replay cannot cope with. In that case, there is limited backward transfer but severe forgetting. In contrast, Adapter+CL and our method demonstrate their robustness after learning more tasks in a stream. Our method also outperforms Adapter throughout the learning process, demonstrating we can enable knowledge transfer even the similarity between tasks is limited.

Case Study We selected e2e in sequence #1 and wiki in sequence #8 as two representative tasks to illustrate the final output generated by different approaches in Table 5. After training on the whole sequence, Adapter+LAMOL cannot correctly con-

vey the information provided in the input, suffering from generating grammar mistakes and missing key points. This could be attributed to the interference from learning new coming tasks. While Adapter+CL successfully mitigate this problem by parameter isolation, our approach works similarly using less parameters and generates better sequences without redundant information.

6.2 The Growth of Compositional Modules

To illustrate the process of adding/reusing modules, we depict the model architecture at each stage in Fig 3 using sequence #4, which is the most challenging sequence containing *similar* tasks according to Table 2. Since the similarity between the second task (e2e) and the first task (hotel) is low (see Figure 4 in Appendix A), our framework automatically learns to add extra adapter modules in layer $\{6, 8, 9, 10, 11\}$ before training on the second task. When the third task (rest) arrives, given its high similarity to the first task, our method correctly decides to reuse all modules used in the first task. Interestingly, the architecture for the fourth task is composed of shared modules with the first 3 tasks in layer $\{1, 2, 3, 4, 5, 7, 12\}$, shared module with the second task in layer 6, shared the mod-

E2E NLG (#1): name[Strada], eatType[coffee shop], area[city centre]	
Reference	*There is a coffee shop in the city centre called the Strada.*
Adapter+CL	*Strada serves coffee, is a nice coffee shop, in city centre.*
Adapter+LAMOL	*Strada is a coffee shop serving city centre food*
Ours	*Strada is a coffee shop located in the city centre.*

WikiSQL (#8): which team has pick 13 in round 2 ?	
Reference	*select team from table where round = 2 and pick = 13*
Adapter+CL	*select team from table where pick = 13 and round = round 2*
Adapter+LAMOL	*select team from table where round = 2 (missing: and pick = 13)*
Ours	*select team from table where pick = 13 and round = 2*

Table 5: Output comparison after training on sequence #1 and #8. We visualized e2e and wiki as two representative tasks and color redundant information in red, missing information in blue and grammar mistakes in orange.

ule with the first and the third task in layer 8, and also added new modules for the fourth task in layer $\{9, 10, 11\}$. For the fifth task, our method reuses all modules used by the fourth tasks due to their high similarity. This demonstrates that our method is adaptive to different incoming tasks and is able to compose modules from different old tasks for new tasks. We also provide a comparison in Appendix B to demonstrate the effect of reusing modules from different transformer layers.

7 Conclusion

This work examined continual sequence generation with adaptive compositional modules, where we proposed hidden state mixing to adaptively compose old and new modules for new tasks and utilized pseudo experience replay to facilitate knowledge transfer. Experiments conducted on various sequence generation tasks demonstrated that our method achieves better performances with higher parameter efficiency over previous state-of-the-art baselines, both on *similar* task sequences and *dissimilar* task sequences. Our work is also subject to a few limitations such as the introduced extra training time. In the future, we plan to investigate how to further speed up the decision stage more efficiently and generalize the current framework to more diverse NLP tasks such as text classification and machine translation.

Acknowledgment

References

Craig Atkinson, Brendan McCane, Lech Szymanski, and Anthony Robins. 2018. Pseudo-recursal: Solving the catastrophic forgetting problem in deep neural networks.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. Multiwoz–a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*.

Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. 2021. An empirical survey of data augmentation for limited data learning in nlp.

Jiaao Chen, Zichao Yang, and Diyi Yang. 2020. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *ACL*, pages 2147–2157.

Yung-Sung Chuang, Shang-Yu Su, and Yun-Nung Chen. 2020. Lifelong language knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2914–2924.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Beyza Ermis, Giovanni Zappella, Martin Wistuba, and Cedric Archambeau. 2022. Memory efficient continual learning for neural text classification.

Junliang Guo, Zhirui Zhang, Linli Xu, Boxing Chen, and Enhong Chen. 2021. Adaptive adapters: An efficient way to incorporate bert into neural machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:1740–1751.

Xu Han, Yi Dai, Tianyu Gao, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2020. Continual relation learning via episodic memory activation and reconsolidation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6429–6440, Online. Association for Computational Linguistics.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Nithin Holla, Pushkar Mishra, Helen Yannakoudakis, and Ekaterina Shutova. 2020. Meta-learning with sparse experience replay for lifelong language learning. *CoRR*, abs/2009.04891.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Yufan Huang, Yanzhe Zhang, Jiaao Chen, Xuezhi Wang, and Diyi Yang. 2021. Continual learning for text classification with information disentanglement based regularization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2736–2746.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does bert learn about the structure of language? In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.

Xisen Jin, Bill Yuchen Lin, Mohammad Rostami, and Xiang Ren. 2021. Learn continually, generalize rapidly: Lifelong knowledge accumulation for few-shot learning. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 714–729, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zixuan Ke, Hu Xu, and Bing Liu. 2021. Adapting BERT for continual learning of a sequence of aspect sentiment classification tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4746–4755, Online. Association for Computational Linguistics.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation.

Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. 2019. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

David Lopez-Paz and Marc' Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Andrea Madotto, Zhaojiang Lin, Zhenpeng Zhou, Seungwhan Moon, Paul Crook, Bing Liu, Zhou Yu, Eunjoon Cho, Pascale Fung, and Zhiguang Wang. 2021. Continual learning in task-oriented dialogue systems. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7452–7467, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Fei Mi, Liangwei Chen, Mengjie Zhao, Minlie Huang, and Boi Faltings. 2020. Continual learning for natural language generation in task-oriented dialog systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 3461–3474.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Mark B Ring. 1998. Child: A first step towards continual learning. In *Learning to learn*, pages 261–292. Springer.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*.

James Smith, Jonathan Balloch, Yen-Chang Hsu, and Zsolt Kira. 2021a. Memory-efficient semi-supervised continual learning: The world is its own replay buffer. *arXiv preprint arXiv:2101.09536*. Accepted for publication at IJCNN 2021.

James Smith, Yen-Chang Hsu, Jonathan Balloch, Yilin Shen, Hongxia Jin, and Zsolt Kira. 2021b. Always be dreaming: A new approach for data-free class-incremental learning. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. 2019. Lamol: Language modeling for lifelong language learning. In *International Conference on Learning Representations*.

Jingyuan Sun, Shaonan Wang, Jiajun Zhang, and Chengqing Zong. 2020. Distill and replay for continual language learning. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3569–3579.

Sebastian Thrun. 1998. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Hong Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang, and William Yang Wang. 2019. Sentence embedding alignment for lifelong relation extraction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 796–806, Minneapolis, Minnesota. Association for Computational Linguistics.

Zirui Wang, Sanket Vaibhav Mehta, Barnabas Poczos, and Jaime Carbonell. 2020. Efficient meta lifelong-learning with limited memory. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 535–548, Online. Association for Computational Linguistics.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Xiuwei Xu, Yifan Wang, Yu Zheng, Yongming Rao, Jie Zhou, and Jiwen Lu. 2022. Back to reality: Weakly-supervised 3d object detection with shape-guided label enhancement.

3663

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, et al. 2019. Learning and evaluating general linguistic intelligence. *arXiv preprint arXiv:1901.11373*.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2017. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A   Supplementary Details and Results

Data and Metric   Table 6 summaries the datasets and metrics we used, all datasets are using the public version from prior work Sun et al. (2019); Chuang et al. (2020)[2]. Note that some big datasets (WikiSQL, CNN/DailyMail, E2E NLG, RNNLG (laptop)) are reduced to a smaller size by random sampling due to data imbalance.

Dataset	Metric	# Train	# Test
E2E NLG		6000	2000
RNNLG(rest.)		6228	1039
RNNLG(hotel)	ROUGE	6446	1075
RNNLG(tv)		8442	1407
RNNLG(laptop)		7944	2649
WikiSQL	lfEM	6525	15878
CNN/DailyMail	ROUGE	6604	2250
MultiWOZ	dsEM	2536	1646

Table 6: Dataset statistics and metrics. Note that ROUGE refers to the mean of ROUGE-1, ROUGE-2 and ROUGE-L, lfEM stands for exact match of logical forms, dsEM represents turn-based dialogue state exact match.

Task Sequences   In the scenario of *CL on dissimilar tasks*, each task sequence also contains two or three similar natural language generation tasks, so the model cannot cheat by always adding new modules without detecting reusable ones.

Implementation Details   We use GPT-2 (Radford et al., 2019) in HuggingceFace Transformers (Wolf et al., 2020) as our backbone. We use the

---

[2]Datasets available at:
https://github.com/chho33/LAMOL
https://github.com/voidism/L2KD

architecture from Houlsby et al. (2019) in Adapter-Hub (Pfeiffer et al., 2020) with its default setting, in which the reduce factor for bottle-neck architecture is 16. All experiments are conducted on NVIDIA RTX 2080 Ti with 11GB memory with a maximum batch size of 4. Training on one task sequence takes 5 to 9 hours.

We use AdamW (Loshchilov and Hutter, 2019) as our optimizer. We select learning rate from $\{1e-4, 1.75e-4, 3e-4\}$ and set the learning rate $lr = 1.75e-4$ for all tasks except WikiSQL, and $lr = 3e-4$ for WikiSQL. For decision stage, we train 6 epochs to make decisions. For training stage, we select the best epoch number from $\{9, 12, 15\}$, and use 9 for *similar* scenario and 12 for *dissimilar* scenario. Weight initialization parameter c is selected from $\{0.03, 0.05, 0.07\}$ for *similar* scenario and $\{0.12, 0.15, 0.17\}$ for *dissimilar* scenario. Loss coefficient γ is selected from $\{0.01, 0.05\}$, η is set to 0.25. Following Sun et al. (2019), we use top-k sampling where $k = 20$ and set the pseudo-data sample rate to 0.2. In our preliminary experiments, increasing the replay frequency can further alleviate forgetting. Thus, for those approaches using pseudo experience replay in this work, we set half of the training batches as pseudo-examples whenever learning a new task.

Note that the original design of Adapter+CL (Madotto et al., 2021) uses perplexity to distinguish which task each testing example belongs to. In this work, we ignore that part and assume that the task-id of each testing example is given during inference for all baselines and our approach to ensure fair comparison.

Finetuning Results   We provide the results of finetuning GPT-2 (Radford et al., 2019) and finetuning adapter (Houlsby et al., 2019) on all eight datasets in Table 7. Since Chuang et al. (2020) shows that the generation loss L_{gen} could slightly increase the performance of finetuning on certain tasks, we also include the finetuning results after adding L_{gen} loss.

Our results confirm that finetuning adapter can almost maintain the performance of finetuning the whole model. We also demonstrated that the performance of finetuning adapter could be improved by simply integrating L_{gen} loss. This suggests that the performance of Adapter+CL could be naively improved by adding L_{gen} to training loss. In that case, the average of mean score for Adapter+CL could be improved to 64.3 on *similar* task sequences and

59.6 on *dissimilar* task sequences, which are still significantly worse than our approach.

Method	e2e	rest	hotel	tv	laptop
GPT-2$_{\text{finetune}}$†	48.8	64.0	65.4	70.8	73.0
GPT-2$_{\text{finetune+gen}}$†	48.8	64.2	65.5	71.0	72.8
Adapter$_{\text{finetune}}$	49.8	64.0	64.9	70.6	71.7
Adapter$_{\text{finetune+gen}}$	49.9	64.3	65.1	70.6	71.8

Method	woz	cnn	wiki
GPT-2$_{\text{finetune}}$†	84.8	25.5	63.1
GPT-2$_{\text{finetune+gen}}$†	82.2	25.9	63.7
Adapter$_{\text{finetune}}$	82.8	26.0	63.1
Adapter$_{\text{finetune+gen}}$	83.5	26.0	63.8

Table 7: Finetuning results, † means we fetch numbers from Chuang et al. (2020)

Figure 4: Task Similarity calculated by the cosine similarity between each task's word frequency distribution.

Results using Geometric Mean   While the mean of all tasks' performance score is always used (Sun et al., 2019; Mi et al., 2020; Madotto et al., 2021) to represent the overall performance on several tasks, it could be largely influenced by the absolute change of one single number. In this work, we also leverage *geometric mean* as an supplementary metric to measure the overall performance on different tasks, which provides another perspective to consider relative change during comparison.

Table 8 summarizes the final performance using geometric mean. We observed the same trend as in Table 2, which demonstrates that our approach does improve the performance of baselines comprehensively on all tasks, not just in favor of absolute value increments on some tasks.

Ablation Study   Table 9 summarizes the full details of ablation study conducted on sequence #1 and #8.

Detailed Final Performance   Table 10 provide the final performance of each task on every sequence for our approach and Adapter+LAMOL. For Adapter+CL, the final results are in Table 7.

Task similarity   Figure 4 illustrates task similarity between five natural language generation tasks, which is calculated by the cosine similarity between each task's word frequency distribution.

B   Module Comparison

In order to demonstrate the compositional nature of our method, that is, each module contains different
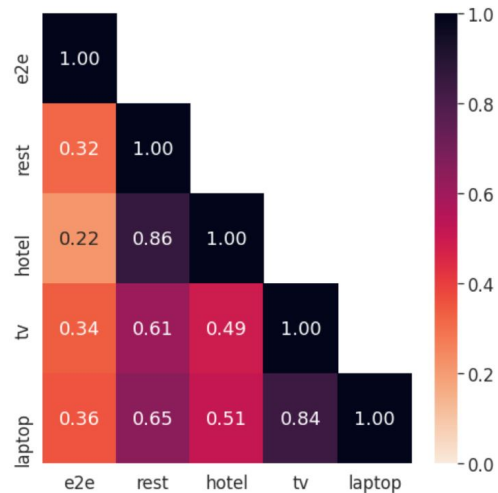
knowledge required for solving each task, we also study the performance difference to quantify the effect of reusing different modules.

Method   After training on task A, we specify a layer $k, k = 1, 2...12$ to add a new module for task B. Then we train the model on task B together with pseudo experience replay. After training on task B, we replace the new module with the old module from task A in layer k, and compare the performance difference on *solving task B* between the modified architecture and the original architecture. On one hand, if the new added module contains specific knowledge of task B, then replacing it will result in the absence of corresponding feature in the generate *output*. On the other hand, if the old module contains specific knowledge of task A, then using it will result in some features of task A being generated in the *output*.

Results   Here we use laptop for task A and e2e for task B. We quantify the task knowledge contained in generated *output* by calculating the cosine similarity of word frequency distribution between specific task's data and generated *output*. In Table 11, we see that replacing the new module in layer 11 results in the most severe information loss of task B in the modified architecture, suggesting that the module in layer 11 contains the most important information of word frequency for task B. In the same way, we conclude that module in layer 3 contains the least important information of word frequency for task B. This is consistent with previous findings (Jawahar et al., 2019) that bag-of-word information

Methods		Finetune	EWC	LAMOL	Adapter +CL	Adapter +Drop	Adapter +LAMOL	Ours
Pseudo Experience Replay		✗	✗	✓	✗	✗	✓	✓
Similar Tasks	# 1	40.2	56.0	65.7	63.7	63.4	65.4	**65.6**
	# 2	35.6	47.9	66.3	63.7	63.4	65.5	**65.8**
	# 3	50.9	60.8	66.0	63.7	63.4	64.9	**65.2**
	# 4	43.1	57.7	66.1	63.7	63.4	64.7	**65.2**
Dissimilar Tasks	# 5	–	–	54.3	53.7	53.4	47.8	**54.6**
	# 6	–	24.0	61.6	64.1	63.6	61.2	**65.0**
	# 7	16.8	36.1	53.4	53.5	52.8	51.3	**54.3**
	# 8	6.62	34.9	53.2	53.5	52.8	47.5	**54.8**

Table 8: Summary of final performance using geometric mean, where "–" denotes no valid geometric mean due to zero. We use two random seeds for each task sequence. Note that the final performance of Adapter+CL and Adapter+Drop is not affected by task ordering within the same group of tasks. For each sequence, we mark the best representation in **bold**, where LAMOL is not compared due to the difference in the order of magnitude of the learnable parameters.

Method	e2e	rest	hotel	tv	laptop	Avg	Avg L.P.	cnn	hotel	wiki	e2e	woz	Avg	Avg L.P.
Ours	51.7	66.7	67.7	72.4	71.9	66.1	2.24M	27.8	65.3	62.9	51.7	83.3	58.2	6.49M
- Entropy loss	52.1	67.1	67.6	72.3	71.5	66.1	2.54M	27.8	64.8	62.6	49.8	82.9	57.6	6.49M
- Weight Ini	49.6	64.7	64.8	70.4	71.3	64.2	7.09M	26.7	64.7	64.6	49.9	82.4	57.7	8.65M
- Pseudo ER	25.6	36.6	39.9	42.8	71.2	43.2	2.08M	23.5	60.2	61.1	50.7	83.9	55.9	6.34M

Table 9: Ablation study on (i) entropy loss (ii) weight initialization (iii) pseudo experience replay. The left part includes results for sequence #1 while the right part includes result for sequence #8. Note that "Avg" refers to the mean of performance score on all tasks and "Avg L.P." refers to the mean of learnable parameters.

is mainly captured by higher transformer layers, while lower transformer layers capture surface and syntactic information.

Similarly, by analyzing the cosine similarity of word frequency distribution to task A, we find that the old module in layer 9 contains the most important information of word frequency for task A and the old module in layer 5 contains the least. While taking a closer look, we also find that modules in different layers contain information of different high-frequency words in task A. For example, module in layer 9, 10 contains the most information of the word "computing", and "laptop", respectively, and module in layer 11 contains more information of the word "business" than any other modules. This further demonstrates that different task-specific knowledge is contained in different modules from different layers, which results in different potential for reuse. By selectively reusing old modules to enable knowledge transfer and adding necessary modules to mitigate knowledge interference, our method derives a **compositional** architecture for every new task, as depicted in Figure 3.

Method - #1	e2e	rest	hotel	tv	laptop	Avg
Adap+LAMOL	51.8	66.5	67.2	72.4	71.5	65.9
Ours	51.7	66.7	67.7	72.4	71.9	66.1

Method - #2	laptop	tv	hotel	rest	e2e	Avg
Adap+LAMOL	74.8	75.2	65.9	66.0	49.3	66.2
Ours	64.7	74.5	51.5	73.5	49.7	66.5

Method - #3	rest	tv	e2e	laptop	hotel	Avg
Adap+LAMOL	64.3	74.9	50.0	74.5	64.1	65.6
Ours	64.7	74.5	51.5	73.5	64.8	65.8

Method - #4	hotel	e2e	rest	laptop	tv	Avg
Adap+LAMOL	66.4	50.9	65.8	73.0	70.0	65.2
Ours	66.4	51.3	66.2	74.2	70.6	65.7

Method - #5	woz	cnn	e2e	rest	hotel	Avg
Adap+LAMOL	75.8	15.4	51.9	64.3	64.3	54.3
Ours	83.5	26.9	51.5	65.1	64.2	58.2

Method - #6	e2e	wiki	hotel	woz	rest	Avg
Adap+LAMOL	53.4	47.9	64.6	80.4	64.7	62.2
Ours	50.9	64.3	65.1	84.1	64.8	65.9

Method - #7	hotel	e2e	woz	wiki	cnn	Avg
Adap+LAMOL	66.0	48.5	77.5	55.4	25.8	54.6
Ours	67.0	50.9	83.5	64.1	25.9	58.3

Method - #8	cnn	hotel	wiki	e2e	woz	Avg
Adap+LAMOL	16.5	65.2	52.5	51.4	83.4	53.8
Ours	27.8	65.3	62.9	51.7	83.3	58.2

Table 10: Final Performance of each task on every sequence. Adap+LAMOL refers Adapter+LAMOL.

Layer	Task A		Task B	
	O	M	O	M
1	59.6	72.5	95.1	92.5
2	60.2	72.3	95.0	93.3
3	60.1	71.3	95.1	_93.6_
4	60.0	70.2	95.1	93.4
5	60.0	_68.9_	95.2	91.3
6	59.8	72.6	95.1	88.3
7	60.0	71.2	95.0	86.2
8	59.9	72.6	95.0	81.9
9	59.6	**76.7**	95.0	83.8
10	59.9	74.1	95.2	81.2
11	59.9	74.5	95.0	**80.3**
12	59.7	75.5	94.9	82.0

Table 11: Module Comparison: the effect of replacing the new module with the old module in different layers after sequentially learning Task A and B. Numbers in this table refer to the cosine similarity of word frequency distribution between *the data of a specific task* and the *output generated from Task B's input* (by original architecture - **O**, or modified architecture - **M**). We highlight the **most informative** layers and the _least informative_ layers differently.