PROGRESSIVE PROMPTS: CONTINUAL LEARNING FOR LANGUAGE MODELS WITHOUT FORGETTING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Progressive Prompts – a simple and efficient approach for continual learning in language models. Our method does not require data replay, and alleviates catastrophic forgetting without using a large number of task-specific parameters. Progressive Prompts learns a new soft prompt for each task and sequentially concatenates it with the previously learned prompts, while keeping the base model frozen. Experiments on standard continual learning benchmarks show that our approach outperforms state-of-the-art methods, with an improvement >20% in average test accuracy over the previous best-preforming method on T5 model. We also explore a more challenging continual learning setup with longer sequences of tasks and show that Progressive Prompts significantly outperforms prior methods.

1 INTRODUCTION

Learning a long sequence of tasks without forgetting remains a key feature of human-level intelligence. While pretrained language models have largely succeeded in finetuning on a single task, their performance degrades in scenarios where multiple tasks are encountered sequentially, also known as *continual learning* (CL) (de Masson D'Autume et al., 2019; Huang et al., 2021). A major challenge in CL is *catastrophic forgetting*, i.e., losing the knowledge acquired from previous tasks after learning new ones (McCloskey & Cohen, 1989; Ratcliff, 1990).

Typical CL approaches for language models use data replay or add a regularization constraint that prevents the model from extensively changing on new tasks (Huang et al., 2021; de Masson D'Autume et al., 2019; Sun et al., 2019). These methods, however, still suffer from forgetting due to inevitable changes in parameters shared between tasks. Other approaches, such as progressive networks (Rusu et al., 2016), can alleviate catastrophic forgetting completely, but are computationally expensive because they add a new copy of the model for each task. This can be especially intractable for large-scale language models with billions of parameters, which have become a standard in the NLP field (Raffel et al., 2020).

In this paper, we introduce **Progressive Prompts** – a novel CL approach for language models. Our method is inspired by progressive networks, but is significantly more memory-efficient because it only learns a fixed number of tokens, or *prompt*, for each new task. Learning a prompt to adapt language models on a single downstream task was introduced in *prompt tuning* (Lester et al., 2021), and was shown to match the performance of full model finetuning while training <0.01% of the parameters. In Progressive Prompts, we learn a separate prompt for each incoming task and sequentially concatenate it with previously learned prompts. Importantly, we share input tokens across all tasks and progressively prepend new prompts while keeping previous prompts frozen (see Figure 1). Our method can: 1) alleviate catastrophic forgetting by preserving the knowledge acquired in previous prompts, and 2) transfer knowledge to future tasks by sequentially learning new prompts given previous ones. We also introduce a new technique for prompt embedding reparameterization (Li & Liang, 2021). We show that by passing the prompt embeddings through a residual MLP we can stabilize prompt tuning and improve its performance.

We run extensive experiments on standard CL benchmarks for text classification, and show that Progressive Prompts outperforms state-of-the-art approaches on both BERT and T5 architectures (Devlin et al., 2018; Raffel et al., 2020). We show over 20% improvement over the current SOTA for T5 model (Qin & Joty, 2021). Furthermore, we run experiments on a more challenging CL setup

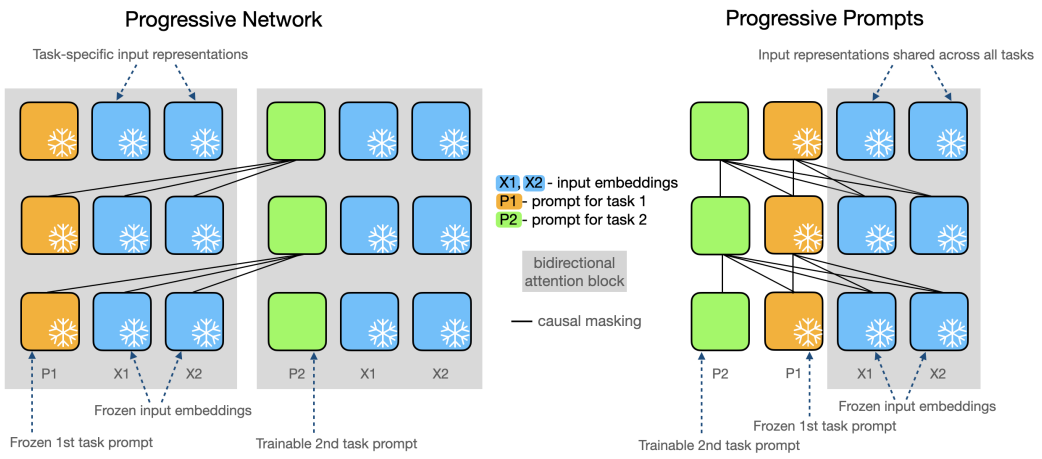

Figure 1: Illustrating our proposed method *Progressive Prompts* and contrasting it with a simple adaptation of progressive networks using prompt tuning. In the simple adaptation of progressive networks we learn a separate prompt and repeat the frozen input embeddings for each new task. This setup requires repeating input tokens for each task. In Progressive Prompts we use the same input and progressively append new prompt for each new task. Prior task prompts are not modified by the addition of new prompts.

with longer task sequences, and show that our method outperforms prior approaches for T5 and BERT architectures.

Our main contributions in this paper are as follows:

- We propose a novel CL approach, *Progressive Prompts*, that successfully alleviates catastrophic forgetting, while learning $< 0.1\%$ of the total parameters. Furthermore, our method allows knowledge transfer to future tasks from previous ones.
- Progressive Prompts is suitable for any transformer-based architecture. We show that it significantly outperforms prior SOTA methods on standard CL benchmarks for both BERT and T5 models.
- We propose a more challenging CL setup encompassing 15 text classification tasks and show that our method significantly outperforms prior methods.

2 Background

2.1 Finetuning

The predominant technique for adapting the language model to a downstream task T is *finetuning*, when all parameters Θ of the model (initialized from some pre-trained weights) are updated during task adaptation (Devlin et al., 2018; Zhang et al., 2020).

Consider a classification task T with input text x, and output scalar label y. Here p_Θ is a probability distribution of output classes parameterized by the weights, Θ, of the language model. In finetuning, we perform gradient updates according to the following log-likelihood objective:

$$\max_{\Theta} \sum_{x,y \in T} \log p_\Theta(y|x)$$

Despite its simplicity and effectiveness, finetuning updates all model parameters, and when there are multiple downstream tasks it requires storing a separate finetuned model for each of them.

2.2 PROMPT TUNING

Recent works explored *prompt tuning* (Lester et al., 2021) as a lightweight alternative for finetuning, when instead of training the whole model, a series of virtual tokens – or *soft prompt* – is trained. In prompt tuning, a soft prompt P is prepended to the input text x, while keeping other parameters frozen. In this case, the model parameters Θ are composed of two sets: 1) the pretrained language model parameters θ which are *fixed* on downstream tuning, and 2) prompt parameters θ_p which are fine-tuned on the given task. Since prompt P has its own dedicated parameters θ_p, we are not restricted by the original model parametrization θ, and can perform gradient updates on the prompt parameters independently.

Given a task T and a probability distribution p_{θ,θ_p} of output classes parameterized by θ and θ_p, the learning objective here is to maximize the log-likelihood of y given the input text x concatenated with the soft prompt P, or

$$\max_{\theta_P} \sum_{x,y \in T} \log p_{\theta,\theta_P}(y|[P;x])$$

2.3 CONTINUAL LEARNING

In this work, we focus on a continual learning setup, where the language model is presented with a sequence of m text classification tasks $(T_1, ..., T_m)$. In each task we have a set if i.i.d. training examples $(x_i, y_i)_{i=1}^{N}$, where x_i is an input text and y_i is a label from a set of predefined labels Y_k associated with the task T_k. We assume that the model is parameterized by Θ and has access to the task identity in both training and inference. Hence, the learning objective across all tasks becomes:

$$\max_{\Theta} \sum_{k=1}^{m} \sum_{x,y \in T_k} \log p_{\Theta}(y|x)$$

The most straightforward approach for continual learning is finetuning, which sequentially optimizes loss for task k, $k \in 1..m$ by updating all model parameters: $\mathcal{L}_k(\Theta) = -\sum_{x,y \in T_k} \log p(y|x, \Theta)$. Continual finetuning will typically result in *catastrophic forgetting*, where performance on the earlier tasks decreases after learning new tasks, and eventually leads to a higher generalization loss (Kirkpatrick et al., 2017; de Masson D'Autume et al., 2019; McCloskey & Cohen, 1989).

3 METHOD

Progressive Prompts We propose *Progressive Prompts*, a continual learning approach which progressively learns a prompt P_k for each new task T_k (Figure 1). With Progressive Prompts we learn a separate prompt P_k for task T_k, and concatenate it with all previously learned prompts $P_i, i < k$, before prepending to the input embeddings. During training, parameters θ of the language model are always frozen, and parameters θ_{P_k} corresponding to prompt P_k are only trainable during learning T_k, and frozen afterwards.

The training objective for task T_k ($k \in \{1...m\}$) is to find prompt parameters θ_{P_k} that minimize the negative log probability of training examples under our progressive prompt and the frozen base model:

$$\mathcal{L}(\theta_{P_k}) = -\sum_{x,y \in T_k} \log p(y|[P_k, ..., P_1, x], \theta, \theta_{P_1}, ..., \theta_{P_k})$$

Such a progressive setup allows to achieve two goals for efficient CL: (1) it successfully eliminates *catastrophic forgetting*, and (2) allows *forward transfer* for subsequent tasks. Since Progressive Prompts trains a separate prompt for each encountered task without modifying its parameters when new tasks are learned, old tasks do not suffer from forgetting. In addition to that, prompts learned on previous tasks allow information re-use for future tasks. A similar phenomenon has been shown by Vu et al. (2021) – prompts learned on informative source tasks served as a good initialization for other downstream tasks.

Embedding reparameterization Li & Liang (2021) have shown that directly optimizing prompt parameters can lead to training instability, while reparameterizing prompt embeddings matrix through

a multi-layer perceptron (MLP) can improve performance. At the same time, Liu et al. (2021) report that prompt embedding reparameterization can hinder performance on certain tasks and is highly sensitive to its initialization. To solve this issue, we propose adding a residual connection to the MLP reparameterization. This residual connection improves optimization, because it avoids learning an identity mapping by the MLP. More specifically, we reparameterize prompt P_k for task T_k as follows:

$$P_k^{'} = \text{MLP}(P_k) + P_k$$

After training on task k is complete, the reparameterization parameters MLP can be discarded, and prompt embeddings P_k can be replaced by their corresponding projection $P_k^{'}$.

4 Experimental Setup

4.1 Datasets

Continual Learning Benchmark We first evaluate our approach on the widely adopted CL benchmark for language models, which includes five text classification datasets by Zhang et al. (2015): AG News (4 classes, news classification), Amazon reviews (5 classes, sentiment analysis), Yelp reviews (5 classes, sentiment analysis), DBpedia (14 classes, Wikipedia text classification) and Yahoo Answers (10 classes, Q&A classification). We provide the task details in Appendix A.1 and sequences of tasks used in our experiments in Appendix A.2.

Following previous works on CL for BERT model, including IDBR (Huang et al., 2021) and MBPA++ (de Masson D'Autume et al., 2019), for BERT-based experiments we use four different orders of these five tasks. We use the same train and test sets as IDBR (Huang et al., 2021) and MbPA++ (de Masson D'Autume et al., 2019), consisting of 115,000 training and 7,600 test examples for each task. Following Huang et al. (2021), for every task we randomly hold out 500 samples per class from the training set for validation, and use early stopping according to the validation accuracy on all seen tasks.

Following CL setup for T5 model as in LFPT5 (Qin & Joty, 2021), we use three different orders of the AG News, Amazon, Yahoo and DBpedia datasets. We replicate Qin & Joty (2021) few-shot CL setting and sample 16 examples per task for the training set and keep the test sets unchanged.

Large number of tasks While previous CL approaches mostly focused on relatively short task sequences of 3-5 tasks (Huang et al., 2021; de Masson D'Autume et al., 2019; Qin & Joty, 2021), a more realistic CL scenario would be long sequences encompassing more tasks. Hence, we create a benchmark of 15 text classification tasks to evaluate the performance of Progressive Prompts along with widely adopted CL approaches. Our benchmark consist of the aforementioned five datasets from CL benchmark, combined with four tasks from GLUE benchmark (MNLI, QQP, RTE, SST2) (Wang et al., 2018), five tasks from SuperGLUE benchmark (Wang et al., 2019) (WiC, CB, COPA, MultiRC, BoolQ), and IMDB movie reviews dataset (Maas et al., 2011). To evaluate performance under different dataset sizes, we create three different versions of each dataset, with 20, 200 and 1000 training samples sampled per class, and report test set performance for each of these settings. Again, we follow Huang et al. (2021) practice and for every task we randomly hold out 500 samples per class from the training set for validation, and use early stopping according to the validation accuracy on all seen tasks. We provide the task details in Appendix A.1 and sequences of 15 tasks used in our experiments in Appendix A.2.

SuperGLUE experiments We also run ablation experiments to understand the effect of forward transfer with Progressive Prompts. We perform experiments on SuperGLUE benchmark (Wang et al., 2019) with Progressive Prompts and compare the performance against original per-task prompt tuning by Lester et al. (2021). Following Lester et al. (2021) setup, we use full training sets of eight SuperGLUE tasks and report best validation performance.

4.2 Baselines

We consider nine baseline methods for comparison with ProgressivePrompts:

- **Finetune** (Wang et al., 2020; de Masson D'Autume et al., 2019; Huang et al., 2021): train all model parameters on a sequence of tasks (without adding any regularization constraints or replaying samples from the previous tasks).
- **EWC** (Kirkpatrick et al., 2017): finetune the whole model with a regularization loss that prevents updating parameters that could interfere with previously learned tasks.
- **A-GEM** (Chaudhry et al., 2018): save examples from the past tasks and restrict the gradients used to update the model on new tasks based on the retrieved examples.
- **Experience replay**: finetune the whole model with a memory buffer, and replay samples from old tasks when learning new tasks to avoid forgetting.
- **MBPA++** (de Masson D'Autume et al., 2019): augment BERT with an episodic memory that saves all seen examples. Perform replay during training, and local adaptation during test time.
- **IDBR** (Huang et al., 2021): BERT-specific approach which continuously trains the whole model using data replay and a regularization loss, which applies sentence representation disentanglement into task-specific and task-generic spaces. Current SOTA on CL benchmark with BERT.
- **Per-task prompts** (Lester et al., 2021): train a separate soft prompt for each task, while keeping the original model frozen. This setup will eliminate catastrophic forgetting, since per-task parameters do not change when new tasks are learned, but will not result in forward transfer.
- **PromptTuning** (Lester et al., 2021; Qin & Joty, 2021): train a shared soft prompt sequentially on all tasks, while keeping the original model parameters frozen.
- **LFPT5** (Qin & Joty, 2021): continuously train a soft prompt that simultaneously learns to solve the tasks and generate training samples, which are subsequently used in experience replay. Current SOTA on CL benchmark with T5.

4.3 Implementation details

Progressive Prompts is a model-agnostic CL method that can be used with any transformer-based model. In our experiments, we use two language models adopted by the previous lines of works in CL for NLP: encoder-only **BERT** model (Devlin et al., 2018) and encoder-decoder **T5** model (Raffel et al., 2020). To compare Progressive Prompts to the recent CL approaches implemented specifically with BERT, we use the pre-trained BERT-base model as in IDBR and MBPA++ methods (Huang et al., 2021; de Masson D'Autume et al., 2019). For comparison with LFPT5 approach (Qin & Joty, 2021) we use pre-trained T5-large[1] model.

BERT Following Devlin et al. (2018), to predict the class of input text x, we use the representation of its first token $h_{[CLS]}$ (which is encoded by a special beginning-of-a-sentence symbol, [CLS]) as a sentence representation. Here h is the whole-input representation matrix from BERT encoder. We apply a linear transformation parametrized by w and a softmax function to obtain the classification probabilities over classes $c \in \{1...\mathcal{C}\}$:

$$p(y = c|h) = \frac{\exp(w_c h_{[CLS]})}{\sum_{y \in \mathcal{C}} \exp(w_y h_{[CLS]})}$$

In BERT experiments we train a separate linear head in addition to prompt embeddings for each task, using cross-entropy loss between the predicted classification probabilities and ground truth class labels.

T5 Following Raffel et al. (2020) and Lester et al. (2021), we adopt text-to-text formulation for all T5 experiments, where classification labels are mapped into words (e.g. 0/1 could be encoded as "True"/"False"). T5 model applies a multi-headed self-attention over the input tokens followed by position-wise feed-forward layers to produce an output distribution over target tokens. We train prompt embeddings for T5 model using cross-entropy loss. We report the rest of the implementation details in Appendix A.3.

[1]Although the original prompt tuning approach reports better performance with T5 v1.1 compared to T5, several subsequent works find version v1.1. less stable for prompt tuning compared to the original T5 and report worse performance (Karimi Mahabadi et al., 2021; Asai et al., 2022). Therefore, in this work we use the original T5 model.

Prompt length For all Progressive Prompt experiments on BERT, we set single-task prompt length to 20 tokens and apply prompt reparameterization with a two-layer residual MLP. For T5 experiments, we use single-task prompt length to 10 tokens in case of long-sequence experiments and 50 in case of T5 experiments (so that total Progressive Prompt length more closely matches LFPT5 prompt length of 300). We report more experiment details in Appendix A.4.

5 Results

For all CL experiments we evaluate methods after training on all tasks and report averaged test set scores across all tasks. Metrics used for different tasks are shown in Appendix A.1. We report Progressive Prompts performance with BERT-base and T5-large models, and compare it to the existing CL approaches used with these models.

5.1 Results on Continual Learning benchmark with T5

		Order			
Method	DR	**1**	**2**	**3**	**avg**
Finetune		18.9	24.9	41.7	28.5
Replay	✓	35.4	37.1	41.5	38.0
EWC		39.0	38.0	44.8	40.6
LFPT5	✓	47.6	52.6	57.9	52.7
ProgPrompt		**75.2**	**75.0**	**75.1**	**75.1**

Table 1: Performance on few-shot continual learning benchmark with T5-based CL approaches. DR denotes whether the method requires data replay. All results are averaged over 3 runs.

Table 1 compares Progressive Prompts performance on the few-shot CL benchmark for T5 model with the existing CL approaches, including previous SOTA – LFPT5 (Qin & Joty, 2021). Notably, Progressive Prompts achieve over 20% improvement compared to LFPT5. This drastic increase in accuracy is explained by the fact that Progressive Prompts method successfully resolves catastrophic forgetting, allowing to significantly improve accuracy on the past tasks.

5.2 Results on Continual Learning benchmark with BERT

Table 2 compares performance of Progressive Prompts across four different task orders with existing CL approaches for BERT model, including previous SOTA – IDBR (Huang et al., 2021). Progressive Prompts further improve performance compared to IDBR, reaching 77.9 average score across all orders. We also note that Progressive Prompts is a model-agnostic approach, contrary to IDBR, which is designed specifically for BERT. Additionally, our method does not require storing data from the previous tasks for future replay, in contrast to the previous commonly used CL approaches for BERT – IDBR and MBPA++.

		Order				
Method	DR	**4**	**5**	**6**	**7**	**avg**
Finetune		14.8	27.8	26.7	4.5	18.4
Replay	✓	67.2	64.7	64.7	44.6	57.8
A-GEM	✓	70.6	65.9	67.5	63.6	66.9
MBPA++	✓	70.8	70.9	70.2	70.7	70.6
IDBR	✓	75.9	76.2	76.4	76.7	76.3
ProgPrompt		**78.0**	**77.7**	**77.9**	**77.9**	**77.9**

Table 2: Performance on continual learning benchmark with BERT-based CL approaches. DR denotes whether the method requires data replay. All results are averaged over 3 runs.

Method ↓		**avg**		**Method ↓**		**avg**	
Num. samples →	20	200	1000	Num. samples →	20	200	1000
T5-Large results				**BERT-base results**			
Finetune	9.7	8.3	7.4	Finetune	31.3	42.4	41.7
Prompt tuning	17.4	13.9	10.9	Prompt tuning	47.6	57.2	59.5
Replay	43.6	44.2	54.4	Replay	36.4	47.0	49.6
Per-task Prompts	69.8	75.2	77.0	Per-task Prompts	50.6	62.4	67.2
LFPT5	54.3	58.2	69.2	IDBR	36.8	47.9	52.2
ProgPrompt	**76.2**	**78.7**	**79.5**	ProgPrompt	**53.5**	**66.9**	**69.3**
MTL	70.7	72.5	76.3	MTL	56.9	67.7	69.9

Table 3: Progressive Prompts outperforms existing continual learning methods on T5-Large and BERT-base models over long sequences of tasks (15 tasks in total). Average results across three different task orders with 20, 200 and 1000 samples per class are shown. MTL denotes multi-task learning.

5.3 Performance with large number of tasks

Table 3 compares common CL approaches, including SOTA methods for T5 and BERT models (Qin & Joty, 2021; Huang et al., 2021), on long-sequence continual learning with 15 tasks. We report averaged results across three task orders (8, 9 and 10) obtained with T5-Large and BERT-base models. We provide non-averaged results for each order in Appendix A.5. To investigate the effect of limited data settings, we perform training on different dataset sizes with 20, 200 and 1000 samples per class. Standard finetuning and prompt tuning suffer from catastrophic forgetting since they update all the model/prompt parameters sequentially, without any constraints, leading to the lowest performance across all CL methods. Previous SOTA approaches, LFPT5 and IDBR, achieve better performance compared to finetuning, prompt tuning and finetuning with data replay, but still suffer from forgetting. Per-task prompt tuning – a method that trains a separate soft prompt for each task – resolves catastrophic forgetting, outperforming LFPT5 and IDBR across all data limits. However, this method cannot achieve transfer between tasks since there is no interaction between prompts of different tasks, in constrast to our approach which encourages knowledge transfer (see Appendix A.5).

Our approach, Progressive Prompts, consistently outperforms all other methods across different data limits, with the strongest improvement in the few-shot setting with 20 samples per class. This can be explained by two factors. Firstly, progressive prompt addition results in *forward transfer* for future tasks, which is especially helpful for low-data regime. Secondly, *residual reparameterization* of the prompt parameters can help to stabilize and accelerate training. We discuss the effect of reparameterization in the next section, and dissect forward transfer between tasks by visualizing attention matrices.

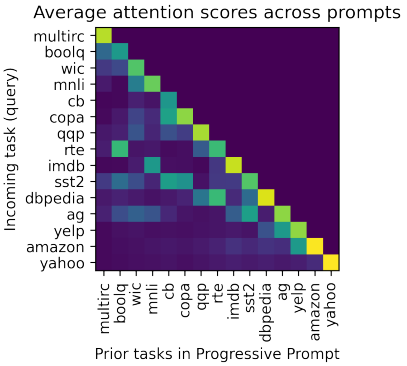

Figure 2: Average attention scores between prompts in Progressive Prompts.

Attention between prompts To investigate which prompts from the progressive pool are helpful for learning new tasks, we visualized between-prompt attentions (Figure 2). We used Progressive Prompts trained on order 8 with T5-Large model. For each of the 15 consecutive tasks, we passed all test examples through T5-Large encoder, and extracted attention matrices from its 24th layer – the last layer of the encoder. We averaged the attention matrices across all examples within the task, and averaged across within-prompt positions, such that each prompt is mapped to a vector of attentions across 16 heads. Finally, to map each prompt to a scalar score we averaged its attention scores across all heads.

Figure 2 shows how a new task prompt, or *query task prompt*, attends to the previously learned prompts, resulting in information reuse from the past tasks. Interestingly, prompts learn to ignore irrelevant tasks and only attend to the most similar or information-rich tasks. For example, prompt corresponding to the 14th task, Amazon reviews (row 14), relies on the preceeding prompt trained on Yelp reviews task, which is very similar to Amazon reviews task. Similarly, Yelp reviews prompt (row 13) strongly attends to the prompts from IMDB reviews and SST2 tasks, which are also sentiment analysis tasks.

5.4 Ablation studies

In this section we dissect two aspects of the Progressive Prompts – *progressive prompt addition* and *residual reparameterization* of prompt embeddings. To quantitatively evaluate effect of forward transfer achieved with Progressive Prompt, we perform experiments on SuperGLUE (Wang et al., 2019) and compare our approach with the original prompt tuning. Also, we explore the improvement obtained from embedding reparameterization.

Improving Prompt Tuning with Progressive Prompts Here we perform experiments on SuperGLUE benchmark to compare the effect of progressive prompt concatenation to the classic prompt tuning. We use T5-Large model for our experiments.

In the original prompt tuning paper, a separate prompt P_T is trained for each SuperGLUE task T (Lester et al., 2021). P_T is a trainable matrix of $100 \times e$ dimension, where 100 is the prompt length and e if the embedding dimension. P_T is concatenated with the input text x and then fed into the model as $[P_T; x]$. For Progressive Prompt we perform training in two steps. First, we learn a progressive prompt on GLUE benchmark P_{GLUE}. P_{GLUE} is composed of 6 per-task GLUE prompts with a length of 10 on random task order, and the final dimension of P_{GLUE} is $60 \times e$. After that, we learn a length-40 prompt $[P_{T_{\text{Prog}}}$ for every SuperGLUE task T and concatenate it with the GLUE prompt before appending to the input x: $[P_{T_{\text{Prog}}}; P_{\text{GLUE}}; x]$ (see Appendix A.6).

Following Lester et al. (2021) we train a separate soft prompt for each SuperGLUE task on its full train set with a fixed number of epochs, and report the validation set performance. Similarly to Lester et al. (2021), for each SuperGLUE task use metrics recommended by Wang et al. (2019) and for tasks with two metrics we compute an average of the two.

Our results are shown in Table 4 – Progressive Prompt improves the standard prompt tuning performance by 2.7 points on SuperGLUE benchmark. This confirms that progressive prompt concatenation allows knowledge reuse from the previous prompts that were learned on GLUE benchmark.

Method	SuperGLUE score
Per-task PT	$74.5_{2.2}$
ProgPrompt	$\mathbf{77.2_{1.9}}$
Per-task FT	$81.3_{0.6}$

Table 4: SuperGLUE performance of the original Prompt Tuning approach vs Progressive Prompts. Averaged validation performance across all tasks is shown.

Prompt len. →	length 5			length 30			
Task ↓	PT	PT+MLP	PT+resMLP	PT	PT+MLP	PT+resMLP	FT
IMDB	$85.8_{0.7}$	$83.4_{0.9}$	$\mathbf{91.4_{0.8}}$	$89.1_{1.0}$	$90.2_{1.5}$	$\mathbf{91.2_{1.4}}$	$92.9_{0.4}$
QQP	$72.9_{0.8}$	$73.1_{1.1}$	$\mathbf{76.6_{1.2}}$	$72.2_{1.5}$	$73.9_{4.1}$	$\mathbf{77.3_{0.9}}$	$78.6_{0.9}$
RTE	$\mathbf{65.0_{2.3}}$	$64.8_{3.2}$	$\mathbf{65.0_{3.5}}$	$\mathbf{67.8_{3.2}}$	$67.6_{3.6}$	$66.2_{3.1}$	$66.2_{2.0}$
MRPC	$72.1_{0.8}$	$71.5_{1.4}$	$\mathbf{75.8_{0.9}}$	$73.4_{0.8}$	$76.9_{3.4}$	$\mathbf{79.0_{1.3}}$	$86.3_{1.6}$

Table 5: Effect of prompt embeddings reparametrization on prompt tuning performance with BERT model. Average performance across 3 runs is shown. **PT**: regular prompt tuning, **PT+MLP**: prompt tuning with prompt passed through 2-layer MLP, **PT+resMLP** (our approach): prompt tuning with prompt passed through 2-layer MLP with a skip connection (*residual MLP*). **FT**: full model finetuning. Best results for each prompt length (5 and 30) are highlighted in bold.

Effect of prompt reparameterization We find that residual reparameterization allows to reach performance close to finetuning, and is especially helpful for BERT model. Table 5 shows the results of regular prompt tuning, prompt tuning with MLP reparameterization and prompt tuning with residual MLP reparameterization on BERT-base model. As in all our experiments, we use a 2-layer MLP with the hidden layer dimension of 800. We show the best performance on four different tasks with prompts of length 5 and 30. Following Lester et al. (2021), we report the maximal validation set performance for each dataset.

Our results show that residual MLP reparameterization results in performance improvement over standard prompt tuning, reaching close to finetuning performance (Table 5). Notably, with length-5 prompt, residual MLP improves accuracy by approximately 6% and 4% for IMDB and QQP datasets, matching full model tuning. Regular MLP reparameterization generally leads to either smaller improvement than residual MLP or even worse performance than prompt tuning.

6 Related work

Continual Learning Existing continual learning approaches can be broadly organized into three main categories: (i) *replay-based*, (ii) *regularization-based*, and (iii) *architecture-based* (de Masson D'Autume et al., 2019; Huang et al., 2021). Replay-based methods store a subset of data from previous tasks for future rehearsal via experience replay, representation consolidation or constrained optimization. The data can be either stored directly or synthesized by generative models. *Regularization-based* approaches restrict changes of model's parameters to avoid inference with previously learned tasks (Li & Hoiem, 2017; Kirkpatrick et al., 2017). *Architecture-based* approaches learn different set of parameters dedicated for a separate task.

Recently, replay-based and regularization-based approaches have been successfully applied for continual learning in language models. While replay-based approaches have shown strong results for classification, question answering and relation extraction tasks, they pose significant memory requirements due to storing large number of samples for rehearsal. Additionally, for many applications such storage would not be feasible due to privacy settings, when access to the past data is not available. Regularization-based approaches are more memory-efficient than replay-based approaches, but suffer from catastrophic forgetting and are often not suitable for long task sequences. In contrast to regularization-based and replay-based approaches, architectural CL approaches are more efficient in resolving catastrophic forgetting problem and, hence, are suitable for sequences spanning a large number of tasks.

Parameter-efficient Learning Recent works on parameter-efficient learning have shown that by training a subset of parameters, we can achieve a full model performance (Houlsby et al., 2019; Li & Liang, 2021; Karimi Mahabadi et al., 2021; Lester et al., 2021). While this line of work has mostly focused on learning a single task, there has been some attempts on using parameter-efficient tuning for CL. For instance, Madotto et al. (2020) proposes AdapterCL which learns a separate adapter block for each task, and Qin & Joty (2021) proposes LFPT5 that learns a large (length 300) soft prompt that is continuously trained on all tasks. Both of these approaches have their limitations – AdapterCL resolves catastrophic forgetting problem, but does not allow forward transfer, while LFPT5 allows forward transfer but suffers from forgetting.

7 Conclusion

This paper presents Progressive Prompts – a novel approach for CL that addresses catastrophic forgetting in pre-trained language models, while allowing knowledge reuse from previous tasks. In contrast to many existing CL methods for NLP, the proposed approach does not require saving examples from the previous tasks for data replay. Moreover, our method does not require storing a large number of task-specific parameters. Progressive Prompts is a model-agnostic approach and our experiments with two commonly used language models demonstrate that Progressive Prompts outperforms baseline methods on a standard CL benchmark for text classification and our custom benchmark of longer CL sequences that spans 15 tasks.

REFERENCES

Akari Asai, Mohammadreza Salehi, Matthew E Peters, and Hannaneh Hajishirzi. Attentional mixtures of soft prompt tuning for parameter-efficient multi-task knowledge sharing. *arXiv preprint arXiv:2205.11961*, 2022.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Cyprien de Masson D'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pp. 2790–2799. PMLR, 2019.

Yufan Huang, Yanzhe Zhang, Jiaao Chen, Xuezhi Wang, and Diyi Yang. Continual learning for text classification with information disentanglement based regularization. *arXiv preprint arXiv:2104.05489*, 2021.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*, 2021.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Andrea Madotto, Zhaojiang Lin, Zhenpeng Zhou, Seungwhan Moon, Paul Crook, Bing Liu, Zhou Yu, Eunjoon Cho, and Zhiguang Wang. Continual learning in task-oriented dialogue systems. *arXiv preprint arXiv:2012.15504*, 2020.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Chengwei Qin and Shafiq Joty. Lfpt5: A unified framework for lifelong few-shot language learning based on prompt tuning of t5. *arXiv preprint arXiv:2110.07298*, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

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:1606.04671*, 2016.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. Lamol: Language modeling for lifelong language learning. *arXiv preprint arXiv:1909.03329*, 2019.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*, 2021.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

Zirui Wang, Sanket Vaibhav Mehta, Barnabás Póczos, and Jaime Carbonell. Efficient meta lifelong-learning with limited memory. *arXiv preprint arXiv:2010.02500*, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*, 2020.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

A Appendix

A.1 Datasets

Table 6 shows details of the 15 datasets we used for our CL experiments, along with their evaluation metrics. Overall, we used datasets from CL benchmark (Zhang et al., 2015), GLUE (Wang et al., 2018) and SuperGLUE (Wang et al., 2019) benchmarks, and added IMDB movie reviews dataset. Following common practive, for tasks that have

two evaluation metrics we use the average of the two as the final performance metric.

Dataset name	Category	Task	Domain	Metric
1. Yelp	CL benchmark	sentiment analysis	Yelp reviews	accuracy
2. Amazon	CL benchmark	sentiment analysis	Amazon reviews	accuracy
3. DBpedia	CL benchmark	topic classification	Wikipedia	accuracy
4. Yahoo	CL benchmark	QA	Yahoo Q&A	accuracy
5. AG News	CL benchmark	topic classification	news	accuracy
6. MNLI	GLUE	NLI	various	accuracy
7. QQP	GLUE	paraphrase detection	Quora	accuracy & F1
8. RTE	GLUE	NLI	news, Wikipedia	accuracy
9. SST2	GLUE	sentiment analysis	movie reviews	accuracy
10. WiC	SuperGLUE	word sense disambiguation	lexical databases	accuracy
11. CB	SuperGLUE	NLI	various	accuracy
12. COPA	SuperGLUE	QA	blogs, encyclopedia	accuracy
13. BoolQ	SuperGLUE	boolean QA	Wikipedia	accuracy
14. MultiRC	SuperGLUE	QA	various	F1 & EM
15. IMDB	Other	sentiment analysis	movie reviews	accuracy

Table 6: The details of 15 datasets used in our CL experiments. NLI denotes natural language inference, QA denotes questions and answers task, EM denotes exact match scoring. First five tasks correspond to the standard CL benchmark, all other tasks are used in our long-sequence experiments.

A.2 Task sequence orders

We report task orders used for our CL experiments across BERT and T5 models in Table 7 below:

Order	Model	Task Sequence
1	T5	db → amazon → yahoo → ag
2	T5	db → amazon → ag → yahoo
3	T5	yahoo → amazon → ag → db
4	BERT	ag → yelp → amazon → yahoo → db
5	BERT	yelp → yahoo → amazon → db → ag
6	BERT	db → yahoo → ag → amazon → yelp
7	BERT	yelp → ag → db → amazon → yahoo
8	T5, BERT	mnli → cb → wic → copa → qqp → boolq → rte → imdb → yelp → amazon → sst2 → dbpedia → ag → multirc → yahoo
9	T5, BERT	multirc → boolq → wic → mnli → cb → copa → qqp → rte → imdb → sst2 → dbpedia → ag → yelp → amazon → yahoo
10	T5, BERT	yelp → amazon → mnli → cb → copa → qqp → rte → imdb → sst2 → dbpedia → ag → yahoo → multirc → boolq → wic

Table 7: Ten different orders of task sequences used for continual learning experiments. Orders 1-7 correspond to the standard CL becnhmark adopted by prior works. Orders 8-10 are our custom long-sequence orders spanning 15 tasks.

A.3 Implementation details

We use PyTorch (Paszke et al., 2019) and HuggingFace Transformers library (Wolf et al., 2019) for our implementation. For the standard CL benchmark, we use official datasets provided by Zhang et al. (2015) available at `http://goo.gl/JyCnZq`, following de Masson D'Autume et al. (2019); Zhang et al. (2015). We use HuggingFace datasets (`https://github.com/huggingface/datasets`) to download data for GLUE tasks (Wang et al., 2018), SuperGLUE tasks (Wang et al., 2019) tasks, and IMDB movie reviews dataset (Maas et al., 2011), which we use for long-sequence CL experiments and/or ablation studies. Following previous studies (Rao et al., 2019; de Masson D'Autume et al., 2019), for CL experiments, for each dataset we use the available validation set as a test set (since test data is not available), and hold out 500 samples from the train set to construct the validation set. For our ablation studies, since we compare Progressive Prompts

with the original prompt tuning (Lester et al., 2021), we follow their set up and report maximal validation set performance.

A.4 Experiment details

We use Adam optimizer (Kingma & Ba, 2014) and set batch size to 8 for all the experiments, except for MTL runs with a batch size of 2 (due to memory limitations). We train each prompt between 10 and 300 epochs, depending on the number of data points. We use the prompt checkpoints with the best validation set score as our final prompts. Prompts are initialized from randomly sampled tokens as in Lester et al. (2021), hyperparametes are shown in the Table 8 below:

Hyperparameter $\downarrow$	**CL benchmark**	**Long-sequence benchmark**		
Num. samples $\rightarrow$	-	1000	200	20
BERT				
Epochs	40	40	150	300
Learning rate	$1e-4$	$1e-4$	$1e-4$	$1e-4$
Prompt length	20	20	20	20
T5				
Epochs	10	10	150	300
Learning rate	0.3	0.3	0.3	0.3
Prompt length	50	10	10	10

Table 8: Hyperparameters used for Progressive Prompts across different CL experiments.

For all CL experiments we use early stopping as in Huang et al. (2021), to save model checkpoint based on the best validation performance on the current task. We report test set performance after training on all tasks as our final metric. For SuperGLUE experiments, we report maximal validation set performance over the course of training as in Lester et al. (2021). We measure the validation performance after every epoch and use metrics described in Appendix A.1. We use 1% of samples per class for the replay approach (but no less than 1 sample per class), following Huang et al. (2021). We use the same hyperparameter setting for all prompt-based approaches (Progressive Prompts, prompt tuning, per-task prompts), except for prompt tuning we use a longer shared prompt of 200 tokens. For all other approaches, we use hyperparameters provided in their corresponding papers.

A.5 Long sequence experiments

Here we report results for orders 8, 9 and 10 of continual learning experiments with long task sequences of 15 tasks. We show test set performance averaged across all tasks for each method. Test scores are calculated after training has been completed. Results are shown in Table 9. We also investigate improvement of Progressive Prompts compared to per-task prompt on the corresponding task in Figure 3. Clearly, some tasks benefit from knowledge sharing from the progressively added prompts.

A.6 SuperGLUE experiments setup

Comparison of the original prompt tuning on SuperGLUE (Lester et al., 2021) and our Progressive Prompt setup is shown in Figure 4.

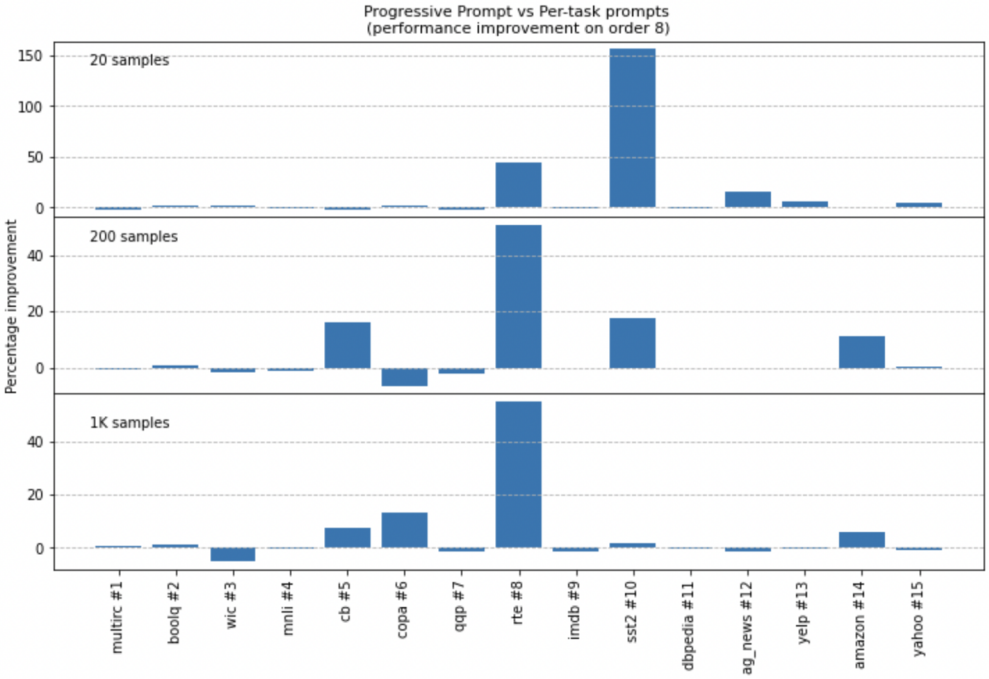

Figure 3: Per-task improvement of Progressive Prompts verus per-task prompts in CL experiment with order 8 across different data limits (20, 200 and 1000 samples per class). X-axis shows the sequence of tasks, Y-axis shows percentage improvement of Progressive Prompts test score compared to per-task prompt on the corresponding task.

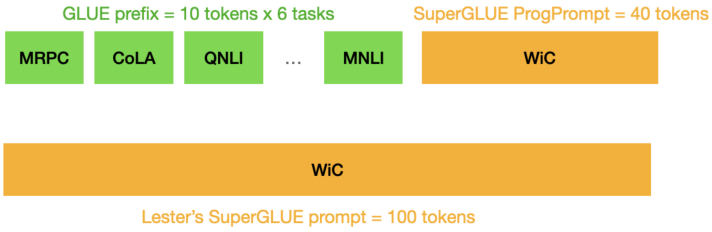

Figure 4: Original prompt tuning versus Progressive Prompts on SuperGLUE datasets. For illustration, we show how SuperGLUE task WiC is leaned (we have similar scheme for other tasks). Prompt tuning trains a single prompt of 100 tokens for WiC task. Progressive Prompts method learns a prompt of 40 tokens, which is progressively appended to the six frozen prompts of 10 tokens learned on GLUE benchmark (with random task order). Total prompt length is equal in both approaches.

Method ↓	**Order 8**			**Order 9**			**Order 10**			**avg**		
Num. samples →	20	200	1000	20	200	1000	20	200	1000	20	200	1000
T5-Large results												
Finetune	9.3	8.9	7.4	9.5	8.1	7.4	10.4	7.9	7.5	9.7	8.3	7.4
Replay	46.0	45.0	55.2	50.3	43.7	54.8	34.6	43.8	53.3	43.6	44.2	54.4
PromptTuning	9.7	8.4	8.2	24.4	16.8	8.7	12.2	8.0	7.9	17.4	13.9	10.9
Per-task Prompts	69.9	75.2	77.0	69.9	75.2	77.0	69.9	75.2	77.0	69.8	75.2	77.0
LFPT5	54.7	61.6	70.4	54.1	54.3	68.2	54.2	58.8	69.1	54.3	58.2	69.2
ProgPrompt	**75.4**	**79.1**	**79.5**	**76.6**	**78.2**	**79.1**	**76.7**	**78.9**	**79.8**	**76.2**	**78.7**	**79.5**
MTL	70.7	72.5	76.3	70.7	72.5	76.3	70.7	72.5	76.3	70.7	72.5	76.3
BERT-base results												
Finetune	29.9	43.4	40.9	30.5	42.0	42.5	33.6	41.9	41.8	31.3	42.4	41.7
Replay	34.9	46.3	51.0	39.3	48.1	51.5	34.9	46.5	46.3	36.4	47.0	49.6
Per-task Prompts	50.6	62.4	67.2	50.6	62.4	67.2	50.6	62.4	67.2	50.6	62.4	67.2
IDBR	39.7	48.4	52.3	37.9	46.6	54.1	32.9	48.8	50.1	36.8	47.9	52.2
ProgPrompt	**55.3**	**67.9**	**68.9**	**53.3**	**65.8**	**70.0**	**51.9**	**66.9**	**69.0**	**53.5**	**66.9**	**69.3**
MTL	56.9	67.7	69.9	56.9	67.7	69.9	56.9	67.7	69.9	56.9	67.7	69.9

Table 9: Average test set performance of Progressive Prompts and common CL approaches on long-sequence experiments with 15 text classicication tasks (orders 8, 9 and 10). We report results for BERT and T5 models across different limits of data – 20, 200 and 1000 samples per class. MTL denotes multi-task learning. All results are averaged over 3 runs.