All Birds with One Stone: Multi-task Learning for Inference with One Forward Pass

Anonymous ACL submission

Abstract

Task-specific fine-tuning of pre-trained language models like Transformers has shown their effectiveness in various NLP tasks. To achieve better storage efficiency and model performance, Multi-task Learning (MTL) has been studied to share model parameters and utilize knowledge transfer between tasks. However, in real applications where enormous numbers of tasks (e.g., large sets of labels to be classified) need to be conducted on a large corpus, the inference efficiency is still hindered by the number of tasks. For a document with N sets of labels to be predicted, recent MTL methods with adaptive modules or prompts need to encode the input data N times to extract the hidden representation needed for the tasks. Notice that the hidden representation is not sharable between tasks, as task-specific features are extracted at very bottom layers in the Transformer. In this paper, we seek to maintain the computational efficiency of only requiring one forward pass for a document to get a generalized feature for all N tasks, without sacrificing overall model performance. We design a prompt-sharing module to let the model take all tasks into considerations and output N heads simultaneously. We also design a dynamic task scheduling module to sample tasks according to their training progress. In our evaluation, we show that our method is able to outperform previous MTL state-of-the-arts and single task fine-tuning by $0.4 - 1.5\%$ on GLUE benchmark dataset. We also perform comprehensive module analysis to demonstrate the effectiveness and robustness of our method.

1 Introduction

Multi-task Learning (MTL) is inspired by the ability of humans to transfer knowledge between tasks. As the prevailing pre-trained langauge models like Transformers (Devlin et al., 2019; Liu et al., 2019b; Clark et al., 2020) are becoming gradually large-sized (Raffel et al., 2020; Brown et al., 2020), it would be storage consuming for researchers to maintain separate fine-tuned models for each downstream task. Thus, the advantage of MTL that shares a large proportion of model parameters between tasks is of great importance. In addition, recent MTL studies develop advanced techniques to achieve good model performance by effectively leveraging knowledge transfer between tasks. Conditional adaptive modules (Pilault et al., 2021) are inserted between layers to extract task-specific features. Prompt-based methods (Lester et al., 2021; Li and Liang, 2021) also output task-specific representations by prepending task-specific prompts to the embedded input.

However, when it comes to real-world applications such as document classification for a large corpus, there can be enormous sets of labels/tasks to be predicted for each document, and efficient inference is extremely important. For a document with N sets of labels to be predicted, recently developed MTL models need to do N forward passes to get the hidden representation for each task. This is because either the adaptive modules or prompts will extract the task-specific features at bottom layers of Transformers and the intermediate features can no longer be shared among tasks. On the other hand, the traditional MT-DNN (Liu et al., 2019a) meets the "one forward pass" requirement by sharing the whole Transformer backbone attached by task-specific linear layers, but the hard-sharing mechanism leads to possible negative transfer, making the model performance lower than adaptive and prompt-based methods.

In this paper, we aim to develop a multi-task learning algorithm that can maintain the inference efficiency to output a generalized representation for all tasks in one forward pass, without sacrificing overall model performance. To achieve this goal, we design a prompt-sharing module that concatenates all task-specific prompts prepended to the embedded input, which ensures that the model

takes all tasks into consideration and outputs N heads simultaneously. However, since such a solution can impair model performance because of negative transfer between tasks, we propose to stop the gradients of other prompts and only backpropagate the gradients for prompts corresponding to the given task being trained. We further add shared prompts to allow for large tasks to expand their prompt capacity. Moreover, training on a set of tasks can be challenging with various task sizes and difficulty. To avoid overfitting to a subset of tasks, we propose to perform dynamic task scheduling. Following (Sharma and Ravindran, 2017), we cast the problem as a multi-armed bandit problem. We design an algorithm based on the Upper Confidence Bound (UCB) solution to probe the model performance periodically to balance over the current metrics and the exploring potential of tasks. Our model is named **MTOP**, for **M**ulti-**T**ask learning for inference with **O**ne forward **P**ass.

To demonstrate the effectiveness of our proposed method, we conduct multi-task experiments on GLUE benchmark (Wang et al., 2018) and show that our method can outperform previous MTL state-of-the-art and single task fine-tuning by $0.4 - 1.5\%$. We also conduct comprehensive module analysis to demonstrate the effectiveness and robustness of our method.

2 Related Work

Multi-task Learning (MTL), which was inspired by the ability of humans to transfer knowledge from previously mastered activities to new ones, has been applied to a wide range of tasks beyond the field of NLP (Caruana, 1997; Ruder, 2017; Donahue et al., 2014). Recent studies in NLP also make successful efforts to train multitasking models to outperform independently fine-tuned ones on the benchmark datasets. MT-DNN (Liu et al., 2019a) uses a shared backbone architecture (e.g., BERT) and task-specific classification heads to optimize corresponding task loss, while BAM (Clark et al., 2019) learns a multi-task model by distilling knowledge from single-task teacher models. With the growing size of language models, light-weight tuning methods are proposed to reduce the number of parameters needed for training a multitask model. Prompt-based methods and adaptor-based methods (Pilault et al., 2021) have shown to achieve higher performance than single task models.

2.1 Prompt-based Methods

Prompts are typically task descriptions or examples prepended to the input of language models. For example, GPT-3 (Brown et al., 2020) takes manually designed prompts as guidance to generate answers for specific tasks. Prompt engineering (Jiang et al., 2020; Schick and Schütze, 2021; Shin et al., 2020) has been studied to search for high-quality templates manually or automatically. However, the non-differentiable nature of discrete tokens can lead to non-optimal results, which motivated the introduction of *soft prompts*, where continuous vectors replace discrete tokens. Prompt tuning (Lester et al., 2021) prepends k tunable soft prompts to input tokens per downstream task. Prefix-tuning (Li and Liang, 2021) prepends continuous task-specific vectors to each layer in the encoder.

2.2 Adapter-based Methods

Adapters are small trainable modules parameterized by task-specific embeddings. These modules are inserted into, or used to replace, key modules to make the transformer model adaptive to new tasks, so that it can project the original rich information to a task-related space. (Houlsby et al., 2019) adds a feed-forward bottleneck and fine-tunes the layernorm of each Transformer layer. CA-MTL (Pilault et al., 2021) remodulates the self-attention layer, normalization layer, embedding projection layer, and bottleneck layer by conditional weight transformation specified by task-specific vectors. By integrating local task modules to the global task agnostic module, (Stickland and Murray, 2019), (Mahabadi et al., 2021), and (Tay et al., 2021) mix efficient adapters to BERT or T5 (Raffel et al., 2020).

2.3 Optimization-based Methods

While the above approaches mostly focus on modifications of model architecture, training on diverse datasets can still be challenging due to possible negative transfer or unbalanced training between tasks. Optimization for MTL is broadly studied to mitigate the issues via gradient modulation and task scheduling. Traditional approaches sample tasks uniformly (Caruana, 1997) or proportionally to data size (Sanh et al., 2018), thus may cause underfitting or overfitting for various task sizes. Subsequent studies have explored dynamic sampling strategies based on the current performance of the model: (Gottumukkala et al., 2020) uses the gap between current metric and the best met-

ric from single task models to reweight the tasks, and CA-MTL (Pilault et al., 2021) uses an entropy-based uncertainty to reweight classification tasks. (Sharma and Ravindran, 2017) studies MTL in reinforcement learning and casts the problem as a multi-armed bandit problem, picking the task with the highest reward in every episode.

Another line of work focuses on eliminating conflicting gradients from multiple tasks. PCGrad (Yu et al., 2020) checks the gradient vectors from multiple tasks and removes the contradictory components if their cosine similarity is smaller than zero. GradVac (Wang et al., 2021) takes a step further to actively update gradients when tasks are positively related. Due to numerous variations of the pairwise gradient modulation between a large number of tasks, in our work we still focus on dynamic task scheduling.

3 Methodology

3.1 Our Prompt Sharing Design

In this section, we will introduce our design of prompt tuning on multi-task problems. Specifically, we will first introduce the preliminary on prompt tuning in Sec. 3.1.1, and then introduce our own design to cope with the multi-task setting in Sec. 3.1.2 and Sec. 3.1.3.

3.1.1 Preliminary on Prompt Tuning

Consider we have a pre-trained Transformer model with pre-trained weights ϕ. A traditional classification model takes a sequence of tokens X with length L_0 as input and takes the output of Transformer at the `[CLS]` token to calculate the class probability $\Pr_\phi(y|X)$. In prompt-based methods, a set of task-specific prompt tokens P with length L are prepended to the input and fine-tuning targets at an optimization of $\arg\max_P \Pr_\phi(y|[P;X])$ with the pre-trained weights ϕ fixed. In GPT-3 (Brown et al., 2020), the prompt tokens P are manually selected from a fixed vocabulary with fixed embeddings. In prompt tuning (Lester et al., 2021), the soft prompt tokens are a new set of parameters $P^e \in \mathbb{R}^{L\times d}$ which can be updated during training, with d being the dimension of hidden representations. Thus we can rewrite the objective as maximizing $\Pr_\phi(y|[P^e;X^e])$, where X^e is the embedding matrix of the input sequence X. While in prompt tuning methods, the pre-trained weights ϕ are often kept fixed in few-shot learning setting, in a multi-task fully supervised setting where tasks enjoy various sizes, difficulty levels and types of loss functions, we found updating ϕ as well as P^e can benefit the final performance and encourage positive transfer among tasks.

3.1.2 Concatenating Prompts from Multiple Tasks

A simple way for fine-tuning N multiple tasks with prompts is to initialize a prompt tensor $P^e_{1:N} \in \mathbb{R}^{N\times L\times d}$, and slice through the first dimension to get $P^e_i \in \mathbb{R}^{L\times d}$, which can be prepended to the input when training on task i, as pictured in Fig. 1(a). This approach is not a good fit for our goal to get multiple task outputs for a document in one forward pass. This is because to achieve the above goal, the hidden representations in a model should be shared among tasks until the very last task-specific classification layers. However, different prompts prepended with the same input sequence will lead to different hidden representations after the first layer of the encoder, thus the following layers can not be shared by multiple tasks as well. To enable hidden representations sharing for multiple tasks, a direct way can be concatenating prompts for different tasks to get a shared prompt pool $\{P^e_1,\dots,P^e_N\} \in \mathbb{R}^{(NL)\times d}$, which is later prepended to the input sequences, as shown in Fig. 1(b). On the output side, the classification layer is no longer attached to the `[CLS]` token, but to an average pooling layer over the hidden outputs of corresponding task prompts. We notate the hidden outputs of the transformer model as $\boldsymbol{H}$, which has the size of $\mathbb{R}^{(NL+L_0)\times d}$, with L_0 as the length of input data. Then the output class probability of task i is calculated by:

$$\boldsymbol{a}_i = \frac{1}{L}\sum_{j=(i-1)L}^{iL-1} \boldsymbol{H}_j \tag{1}$$

$$Pr_\phi(y_i) = \text{Softmax}\left(V_i\,\sigma\left(W_i\boldsymbol{a}_i + \boldsymbol{b}\right)\right) \tag{2}$$

where $\boldsymbol{H}_j$ is the j-th sliced vector along the first dimension of $\boldsymbol{H}$. Notice that different from using the `[CLS]` token to be sent into the classification layer, leveraging the hidden outputs of corresponding prompts can be conceptually seen as using different `[CLS]` tokens for different tasks.

3.1.3 Gradient Stopping for Sharing Prompts

The above direct way of sharing prompts does not lead to a good performance, which we will show in

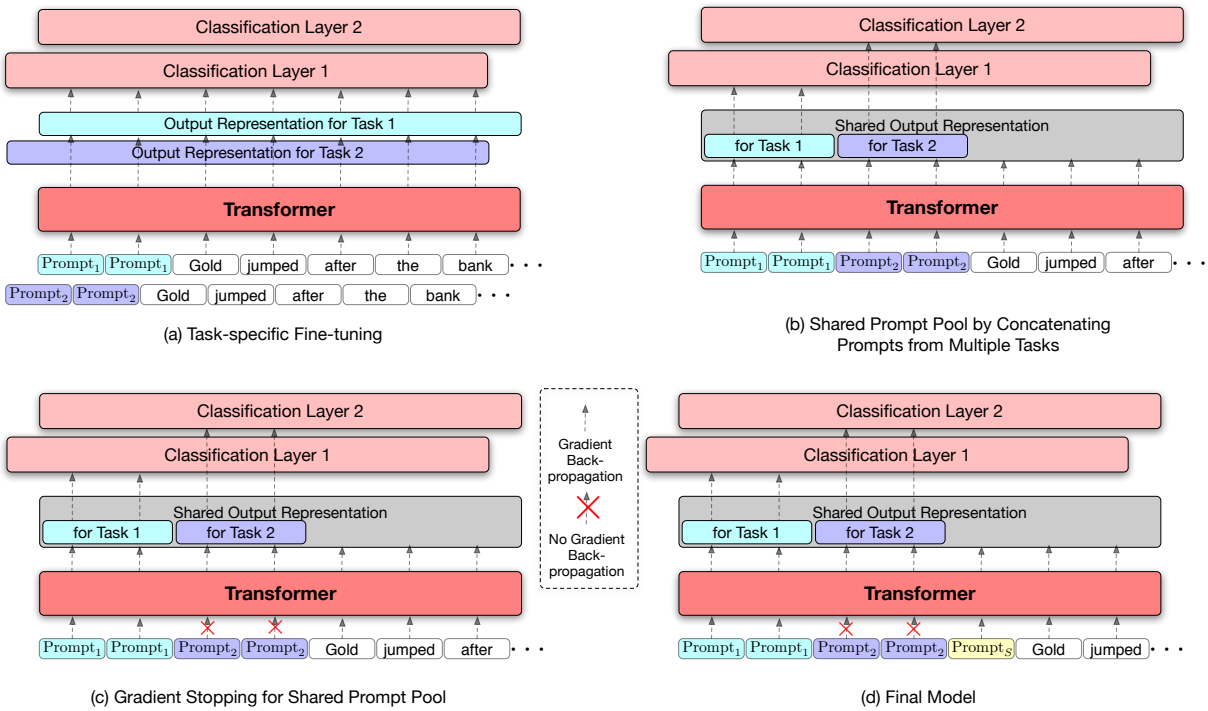


Figure 1: Paradigms of multi-task prompt tuning. (a) Task-specific prompts are prepended separately to the input and the Transformer outputs separate representations for different tasks. (b) The prompts for multiple tasks are concatenated together and prepended to the input. The output hidden representation of Transformer are shared by classification layer of all tasks. (c) The output hidden representation of Transformer are shared by classification layer of all tasks. However, there is only gradient backpropagation for prompts corresponding to the task being trained. (d) Our Final Model: Adding shared prompt tokens to (c). This allows the model the flexibility to expand prompts for larger tasks.

our experiments in Sec. 4.4. This is due to possible negative transfer between tasks. For example, when training a batch of data in task i, backpropogation not only updates the corresponding prompt embedding P_i^e, but also affects irrelevant prompt embeddings $P_j^e (j \neq i, 1 \leq j \leq N)$.

Given the above concern, we only allow the gradients of task i to backpropagate to prompt embedding P_i^e, and stop the gradients for other prompts, as shown in Fig. 1(c). The gradient-stopping prompt sharing benefits the performance of the model in two aspects: (1) Multiple prompts are input together into the self-attention modules so related tasks are easy for the Transformer model to identify and refer to; and (2) non-related tasks will less likely impair the results of each other due to the gradient stopping policy. We notate the shared prompt pool with gradient stopping policy as $\widetilde{P}_S^e = \{\widetilde{P}_1^e, \ldots, \widetilde{P}_N^e\} \in \mathbb{R}^{(NL)\times d}$.

In our experiments, we empirically discover that combining $\widetilde{P}_S^e$ with another shared prompt $P_S^e \in \mathbb{R}^{L'\times d}$ leads to the best performance, as shown in Fig. 1(d). Our reasoning is that the optimal length of prompts varies by task sizes, and the shared prompts allow the flexibility for large tasks to expand their prompt capacity. In our final model, the input to the text encoder is $[\widetilde{P}_S^e; P_S^e; X^e]$ and the output probability of task i is derived by Eq.(1) and Eq.(2).

3.2 Dynamic Task Scheduling

Prior studies on multi-task learning typically sample tasks uniformly or proportionally to task size, while some recent studies (Gottumukkala et al., 2020; Pilault et al., 2021) adopt dynamic sampling strategies to select the most needed task during model training. In our paper, we refer to a study (Sharma and Ravindran, 2017) in Reinforcement Learning (RL) and cast the problem as a multi-armed bandit problem with discounted rewards. We first introduce the preliminary on multi-armed bandit problem and its discounted version in Sec. 3.2.1 and then introduce our own adaptations in Sec. 3.2.2.

3.2.1 Preliminary on Multi-armed Bandit Problem

The multi-armed bandit problem aims to reach the maximum expected reward by a strategy to select a sequence of actions/arms. This can be comparable to our goal of reaching the optimal performance within a fixed amount of training steps on different tasks. The Upper Confidence Bound (UCB) algorithms (Auer et al., 2004; Auer and Ortner, 2010) are often used for solving bandit problems, which balances between exploitation and exploration of arms. At time step t, the action taken can be selected by:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{n_t(a)}} \right] \quad (3)$$

where $Q_t(a)$ is the exploitation reward of action a and the second term indicates the exploration bonus of a, which decreases as $n_t(a)$ (the number of times action a has been taken) accumulates.

To apply the algorithm to multi-task learning, (Sharma and Ravindran, 2017) regards A_t as the task chosen at step t and relates $Q_t(a)$ to the performance (e.g., F1-score on dev set for classification problems) of task a at step t. They further point out that different from stochastic multi-armed bandit problems that have stationary rewards, in multi-task learning, the more times a task is chosen, the less reward that task gains. A discounted UCB algorithm is applied in their study which discounts $Q_t(a)$ and $n_t(a)$ by a factor γ as the training step grows.

3.2.2 Our Adaptations

Directly applying the discounted UCB algorithm to our setting is problematic for two reasons: (1) Unlike (Sharma and Ravindran, 2017) that calculates reward $Q_t(a)$ as the gap between a target score and the current score, we are unable to acquire a suitable "target" score for each task unless performing single-task fine-tuning in advance; and (2) their study updates $Q_t(a)$ at every step, but obtaining a metric value on dev set at every training step can be prohibitively expensive.

In our dynamic task scheduling algorithm, we define a metric probing interval I to update $Q_t(a)$. At time step $t = K \cdot I$, we update the exploitation reward $Q_t(a)$ for each task a to be the sum of its discounted history and the improvement over I steps, divided by the discounted time steps trained on a:

Algorithm 1: Dynamic task scheduling.

Input: Pre-trained model M_{MUL} and N tasks to be trained in a multi-task setting. Metric probing interval I, discounted factor γ, exploration weight c.
Output: A fine-tuned model M_{MUL} on multiple tasks.

```
for a ← 1 to N do
    Q_0(a) ← 0;
    n_0(a) ← 0;
//Train for T steps;
γ_q ← γ^I;
for j ← 0 to T/I − 1 do
    if j > 0 then
        for a ← 1 to N do
            Q_t(a) ← Eq. (4);
    for k ← 1 to I do
        Select task A_t by Eq. (6);
        for a ← 1 to N do
            n_t(a) ← Eq. (5);
        Train M_MUL on task A_t for 1 step;
```

$$Q_t(a) = \frac{\gamma_q Q_{t-I}(a) + \Big(m_t(a) - m_{t-I}(a)\Big)}{n_t(a)} \quad (4)$$

where γ_q is the discounted factor for $Q_t(a)$, and $m_t(a)$ is the metric value of model at step $K \cdot I$ on dev set. In experiments we will demonstrate our design of using performance progress between I steps is better than using the gap between the target metric and current metric as proposed in (Sharma and Ravindran, 2017).

Between step $K \cdot I$ and $(K+1) \cdot I$, we keep $Q_t(a)$ fixed but update the discounted number of steps $n_t(a)$. When task A_t is selected, $n_t(a)$ is updated as:

$$n_t(a) = \gamma n_{t-1}(a) + \mathbb{1}(a = A_t) \quad (5)$$

We set $\gamma_q = \gamma^I$ to keep the same discounting progress inside and outside the I steps. Our task selection strategy is as below:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log \sum_a n_t(a)}{n_t(a)} \cdot w(a)^\alpha} \right] \quad (6)$$

Notice that we modify the exploration term with $w(a)$ being the size of task a, indicating that the exploration term samples tasks proportionally to their sizes when $\alpha = 1$. To avoid catastrophic forgetting on small tasks, we refer to the "annealed sampling" in (Stickland and Murray, 2019) to train on tasks more equally towards the end of training, and set α to gradually decrease by:

$$\alpha = 1 - 0.8\frac{e-1}{E-1} \quad (7)$$

Methods	COLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	AVG
Single Task	63.2	88.4/87.9	89.2	92.9	72.3	69.6	95.7	88.3	82.4
MT-DNN	62.3	88.1/87.2	87.2	92.8	72.4	80.6	95.2	88.8	83.4
Prompt Tuning	60.6	87.8/87.0	89.4	93.0	72.0	80.8	95.2	88.9	83.4
P-Tuning v2	60.2	87.8/86.9	88.6	92.7	72.5	79.2	94.2	89.2	83.0
CA-MTL	61.2	87.7/87.2	89.0	92.7	76.0	79.0	94.2	88.5	83.5
MTOP (w/o DTS)	63.6	87.7/87.4	88.9	92.7	72.1	80.8	95.2	89.1	83.7
MTOP	65.1	87.4/86.6	89.4	92.7	72.0	81.1	95.2	88.5	**83.9**

Table 1: Performance of all methods on test set of GLUE benchmark. Reported metrics: F1 scores for QQP/MRPC, spearman's correlation for STS-B, matthew's correlation for CoLA and accuracy for other tasks. **MTOP** is our complete method and **MTOP (w/o DTS)** is our proposed method without Dynamic Task Scheduling (DTS).

where e is the current epoch number and E is the total number of training epochs. An overall dynamic task scheduling algorithm is described in Algorithm 1.

4 Experiments

In this section, we demonstrate the effectiveness of our proposed method **MTOP** on the GLUE benchmark dataset (Wang et al., 2018). We also provide a series of module analysis to evaluate the performance of each module.

4.1 Experiment Setup

We use ELECTRA-base (Clark et al., 2020)[1] as the backbone model for our method and baselines, and set the maximum sequence length to be 128. We choose the best hyperparameters based on the model performance on GLUE dev set, and then use the best model to predict on GLUE test set. The hyperparameters are listed below: For prompt sharing, we use 4 task-specific prompt tokens per task in GLUE and 32 shared prompt tokens. For dynamic sampling, we obtain the model performance at each $I = 1000$ step. We set the discounted factor $\gamma = 0.99$ and the exploration weight $c = 2\text{e}{-5}$. For dataset training, we use a batch size of 16 for each task, and let the model train for 5 epochs. For optimization, we set the learning rate to be $1\text{e}{-5}$ and use linear warm-up schedule.

4.2 Compared Methods

We compare with several important multi-task learning baselines and recently proposed prompt-tuning methods, as well as the single-task fine-tuning methods. For all baselines, we load ELECTRA-base model as the encoder. The compared methods are listed below:

- **Single Task**: Single task fine-tuning on each GLUE task independently.
- **MT-DNN** (Liu et al., 2019a): A multi-task deep neural network that uses the transformer encoder as a shared architecture among tasks, and the top layer is attached to task-specific linear layers.
- **Prompt Tuning**: The vanilla prompt tuning in (Lester et al., 2021) where task-specific prompts are trained on each task.
- **P-Tuning v2** (Liu et al., 2021): A recently proposed advanced deep prompt tuning method that adds independent prefix tokens in each transformer layer.
- **CA-MTL** (Pilault et al., 2021): The current state-of-the-art model for multi-task learning that uses adaptive modules parameterized by task embeddings.

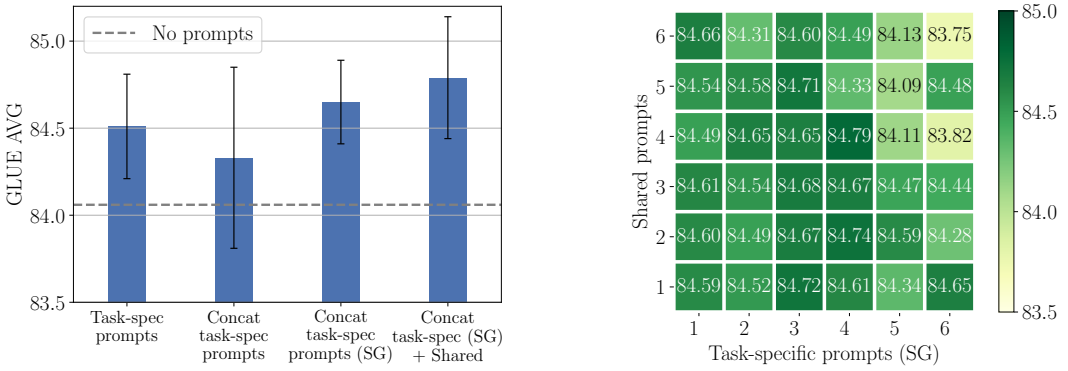
For our own method, we report on two variations:

- **MTOP (w/o DTS)**: Our proposed prompt-sharing module without Dynamic Task Scheduling (DTS), only sampling tasks proportionally to their sizes.
- **MTOP**: Our complete proposed method with prompt sharing and dynamic task scheduling.

4.3 Evaluation on GLUE

We evaluate our proposed model **MTOP** as well as other baseline methods on GLUE test set of 8 tasks in Table 1. As shown in the table, the multi-task learning methods have higher average score than the single task fine-tuning method, indicating the

[1]There are two base ELECTRA models trained from different datasets. Here we use the base++ version derived from the XLNET dataset.



(a) Performance of different prompt sharing settings on GLUE dev set.

(b) Performance of models on GLUE dev set with different numbers of prompt tokens per task.

Figure 2: Ablation and hyperparameter studies on prompt sharing module (without dynamic task scheduling). Mean and Standard deviation are obatined from 15 runs.

effectiveness of sharing the same backbone architecture between tasks. **MTOP (w/o DTS)** performs better than other prompt-based methods, showing that our proposed prompt-sharing module extracts better task-related features. The CA-MTL method is the current state-of-the-art on multi-task learning, but it overfits too much to the QQP dataset and is not better than ours on the average score. This is because they calculate classification uncertainty based on shannon entropy to sample the tasks, thus weighing QQP dataset too much due to its F1-score being around 70 (compared to other co-trained tasks with metrics around 80 or 90). Among all the methods, **MTOP** achieves the best average score, and performs especially well on smaller tasks (COLA, MRPC and RTE). This shows that smaller tasks benefit more from multi-task learning, and dynamic task scheduling further improves the results by balancing between large and small tasks. As the trade-off, we notice metric for larger task like MNLI slightly drops. We consider there is still headroom for the proposed multi-task model to improve.

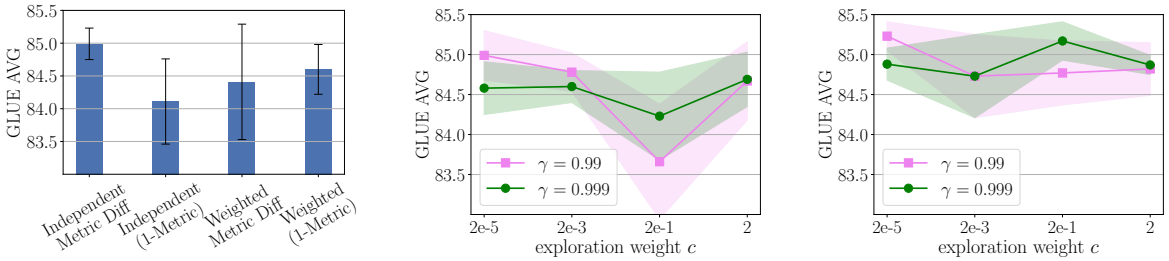
4.4 Analysis on Prompt Sharing Module

To analyze the prompt sharing module, we disable the dynamic scheduling and contrast over five different prompt settings: (1) using no prompts and only do task-specific fine-tuning as a baseline; (2) Fig. 1(a): using the prompt-tuning (Lester et al., 2021) method that prepends task-specific prompts to corresponding inputs; (3) Fig. 1(b): concatenating all task-specific prompts to be shared by all inputs; (4) Fig. 1(c): concatenating task-specific prompts for all inputs but stopping the gradient for prompts of other tasks; and (5) Fig. 1(d): combining gradient-stopping task-specific prompts with shared prompts. We show the performance on GLUE dev set under these settings in Fig. 2(a). The results clearly shows that concatenating the task-specific prompts only does not help knowledge transfer between tasks and can even harm the original prompt-tuning method. However, if we allow the gradient of a task to only propagate back to its own prompts and stop updating the other prompts, the results can be better than prompt-tuning. Adding shared prompts further improve the results, since shared prompts allow the flexibility for large tasks to expand their prompt capacity.

We perform hyperparameter study for prompt-sharing module by varying the length of task-specific/shared prompt tokens, and show the model performance in Fig. 2(b). Overall, the model performs best when the number of task-specific and shared prompt tokens per task are 4 (32 prompt tokens on 8 GLUE tasks), and is quite stable when the prompt length is even smaller. This indicates that our proposed method is robust and does not require a large set of new parameters.

4.5 Analysis on Dynamic Task Scheduling

For dynamic task scheduling, we first demonstrate four options for updating the exploitation reward $Q_t(a)$ and their performance on GLUE dev set in Fig. 3(a). 'Metric Diff' is our choice that calculates the performance gap between I steps, while '(1-Metric)' indicates the gap between target performance (set to 1) and current performance. 'Independent' means that the exploitation reward $Q_t(a)$ for each task is calculated independently accord-



(a) Performance of using different exploitation rewards in dynamic task scheduling ($\alpha = 1$).

(b) Hyperparameter study on exploration bonus weight c in dynamic task scheduling ($\alpha = 1$).

(c) Hyperparameter study on exploration bonus weight c in dynamic task scheduling ($\alpha = 1 - 0.8\frac{e-1}{E-1}$).

Figure 3: Hyperparameter study on dynamic task scheduling module. Mean and Standard deviation are obatined from 15 runs.

ing to Eq. (4), while 'Weighted' means we add up $Q_t(a)$ for all tasks and distribute the total reward to tasks by their task sizes. Fig. 3(a) clearly shows that our proposed adaptations of task scheduling is more effective and stable than traditional approaches.

We also perform hyperparameter for dynamic task scheduling by varying the discounted factor γ and the exploration weight c. We illustrate the results of proportional sampling by exploration bonus ($\alpha = 1$) in Fig. 3(b) and annealed sampling by exploration bonus ($\alpha = 1 - 0.8\frac{e-1}{E-1}$) in Fig. 3(c). We observe that model performance is more stable under the annealed sampling than proportional sampling, and the best exploration weight is around 2e$-$5.

4.6 Discussions

We discuss about the effect of dynamic task scheduling. In Table 2 we list the average scores on GLUE dev set and test set for three different task scheduling settings: (1) **w/o DTS** which purely uses proportional sampling; (2) **w/ DTS** ($\alpha = 1$) which lets the explorations bonus term weigh tasks proportionally, and (3) **w/ DTS** ($\alpha = 1 - 0.8\frac{e-1}{E-1}$) which lets the explorations bonus term weigh tasks from proportionally to equally. We found that even though both DTS strategies perform better on GLUE dev set than without DTS, their GLUE test set performances are not always better. We believe this is because DTS uses dev set scores in the training process to adjust task weights, thus the optimal hyperparameters might overfit to dev sets, and cannot achieve the best performance on the test set. In the future we will explore more on improving the dynamic task scheduling technique such as partitioning the training set to get held-out datasets and using ensemble methods to simulate the rewards.

Ablations	GLUE dev	GLUE test
w/o DTS	84.79 (0.24)	83.74
w/ DTS		
$\alpha = 1$	84.99 (0.31)	83.45
$\alpha = 1 - 0.8\frac{e-1}{E-1}$	85.23 (0.18)	83.88

Table 2: Average scores on GLUE dev set (with standard deviation over 15 runs) and test set of different dynamic task scheduling settings.

5 Conclusion

In this paper we propose an inference-efficient multi-task learning algorithm that is able to output a generalized representation for all tasks in one forward pass, while preserving the overall model performance. We propose a prompt-sharing module with gradient stopping to let the model take all task prompts into consideration and output all task heads simultaneously. To further improve the results, we propose a dynamic task scheduling method by probing the model performance and adapting an algorithm for multi-armed bandit. Our method achieves strong performance on GLUE benchmark dataset and outperforms previous state-of-the-art.

Several interesting directions can be explored in future studies. Based on our prompt-sharing module, we can add MOE layers or other prompt merging techniques to deal with scalable issues. For the dynamic scheduling module, other UCB-based algorithms are worth exploring. Moreover, since our architecture is focused on sentence-level tasks, it would be interesting to extend the study to token-level tasks where each token has a set of labels to be predicted.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2004. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256.

Peter Auer and Ronald Ortner. 2010. Ucb revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61:55–65.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019. Bam! born-again multi-task networks for natural language understanding. In *ACL*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. 2014. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*.

Ananth Gottumukkala, Dheeru Dua, Sameer Singh, and Matt Gardner. 2020. Dynamic sampling strategies for multi-task reading comprehension. In *ACL*.

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 *ICML*.

Zhengbao Jiang, Frank F. Xu, J. Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *EMNLP*.

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

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

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019a. Multi-task deep neural networks for natural language understanding. In *ACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*.

Jonathan Pilault, Amine Elhattami, and Christopher Joseph Pal. 2021. Conditionally adaptive multi-task learning: Improving transfer learning in nlp using fewer parameters & less data. In *ICLR*.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *ArXiv*, abs/1706.05098.

Victor Sanh, Thomas Wolf, and Sebastian Ruder. 2018. A hierarchical multi-task approach for learning embeddings from semantic tasks. In *AAAI*.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *EACL*.

Sahil Sharma and Balaraman Ravindran. 2017. Online multi-task learning using active sampling. In *ICLR*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Eliciting knowledge from language models using automatically generated prompts. In *EMNLP*.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *ICML*.

Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2021. Hypergrid transformers: Towards a single model for multiple tasks. In *ICLR*.

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

Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. 2021. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *ArXiv*, abs/2010.05874.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. In *Neurips*.