Parameter-Efficient Adaptation of BERT using LoRA and MoE

Stanford CS224N Default Project

Li-Heng Lin
Department of Computer Science
Stanford University
`lhlin@stanford.edu`

Yi-Ting Wu
Department of Electrical Engineering
Stanford University
`yiting31@stanford.edu`

Abstract

As pre-trained models grow larger, fine-tuning the entire model specifically for every downstream task becomes impractical. Our project explored a parameter-efficient technique, Low-Rank Adaptation (LoRA), for fine-tuning pre-trained models. Furthermore, we investigated different weight sharing approaches to see if we can achieve even better performance from multitask learning. Specifically, we proposed a procedure to integrate a popular architecture, Mixture-of-Experts (MoE), into the pre-trained BERT weights. We found out that LoRA is able to achieve competitive or even better results than full-model fine-tuning, but we cannot improve the performance further by using more sophisticated weight sharing schemes. We hypothesize stronger regularization is needed to make use of more complex model architectures, but leave it for future work.

1 Key Information to include

- Mentor: Olivia Lee
- External Collaborators (if you have any): None
- Sharing project: No
- **Late Days: 3** (total of 6 days shared between two team members)

2 Introduction

Fine-tuning large pre-trained model has shown to be an effective way to obtain performant models on various downstream tasks. Full-model fine-tuning, a common technique that retrains all model parameters, delivers decent multi-task generalization. Techniques like Pre-Finetuning by Aghajanyan et al. (2021) introduce additional pre-training stages specifically designed to enhance performance across diverse downstream tasks. However, these advanced techniques, despite their effectiveness, often require significant computational resources and may not scale well with the ever-increasing size of pre-trained models. To address this challenge, parameter-efficient fine-tuning techniques like Low-Rank Adaptation (LoRA) by Hu et al. (2021) have emerged. By leveraging the low-rank properties of transformers, LoRA enables fine-tuning with considerably fewer parameters.

This paper first investigates the efficacy of LoRA in achieving performance comparable to full-model fine-tuning. We found out that learning a task-specific scaling variables provides non-trival performance improvement, implying every task requires a different amount of change to the model weights. With this design, our model is able to achieve competitive or even better performance than full-model fine-tuning.

We also explore various ways to learn transferrable knowledge, hoping to get benefits from multitask learning. Specifically, we tried to add a designated LoRA matrix for learning shared knowledge and propose a novel way to integrate MoE layers into pre-trained BERT architecture. However, we cannot push the performance further beyond separately fine-tuning BERT for each task using LoRA. We found out that with task-specific LoRA matrices, the model already fits the training data almost perfectly, so the bottleneck is how to close the train-dev gap instead of designing better architecture to extract more knowledge from training data. We hypothesize stronger regularization such as SMART (Jiang et al. (2019)) might help but leave it for future work.

3 Related Work

3.1 Parameter-Efficient Fine-Tuning

Parameter-efficient fine-tuning (PEFT) tackles the high computational cost of fine-tuning large models by training only a subset of parameters. The Adapter module by Houlsby et al. (2019) reduces training parameters by freezing the pre-trained model backbone and inserting small, lightweight layers that are further trained for specific tasks. AdapterFusion by Pfeiffer et al. (2020) builds upon this concept by incorporating the Adapter module into a two-phase architecture for separate knowledge extraction and composition. This allows classifiers to leverage representations learned from multiple tasks without introducing new parameters, maintaining a low memory footprint.

However, these methods with additional layers can increase inference time. LoRA (Low-Rank Adaptation) addresses this by reducing parameter size without impacting inference latency (Hu et al. (2021)). LoRA demonstrates that decomposing transformer attention matrices into low-rank matrices enables efficient fine-tuning by training only the low-rank parameters while maintaining accuracy. DoRA (Dynamic Low-Rank Adaptation) proposed by Liu et al. (2024) inherits the LoRA structure but further decomposes pre-trained weights into magnitude and directional components to enhance learning capacity and training stability.

While these studies focus on adapting low-rank architectures to a single task, MultiLoRA by Wang et al. (2023) and LoraHub by Huang et al. (2023) explore scaling LoRA modules horizontally and merging different LoRA matrices for more diverse applications such as cross-task and unseen task generalizations. This paper further investigates different variations of LoRA models and how they adapt in multi-task applications.

3.2 Mixture-of-Experts

Proposed by Shazeer et al. (2017), Mixture-of-Experts (MoE) layer has recently been a popular architectural choice to scale up transformer-based Large Language Models. Each MoE layer consists of a router and a set of experts, and the router determines which experts would be used to process a specific token. A core to this architecture is the gating network. In the original work by Shazeer et al. (2017), they introduced a noisy top-k gating, where the sparsity saves computation and the noise helps with load balancing. Fedus et al. (2021) further simplified MoE to top-1 gating, achieving 4x pre-training speed-up on a 1.6T parameter model. Instead of routing token to the expert, there is also work like Soft MoE (Puigcerver et al. (2024)) that lets the expert choose a mixture of inputs to process.

While the major focus of the MoE layer recently is to keep the FLOPs per token roughly constant while increasing the number of parameters by a large amount, in this project, we view it more like a parameter-efficient alternative to ensembling. We are curious about whether the router and experts design can help the model to learn specialized experts and apply them intelligently for different tokens.

4 Approach

We organize this section in the following way. First, we'll describe the output head parameterization to adapt BERT embeddings to different tasks. Then, we'll talk about how we design the architecture based on BERT to learn multiple tasks in a parameter-efficient way. Finally, we'll describe the loss function to train the model on multiple tasks.

4.1 Head Parameterization and Task-Specific Loss Function

We only show the choice we use in our final model in this section. We refer the reader to Section 5.3 for different options we tried.

Sentiment Analysis We use a linear layer to turn BERT's pooler output to a 5-class logits. Then, we use a softmax operation to turn the 5-class logits into a valid probability distribution. We use cross entropy loss to train this head.

Paraphrase Detection We separately pass the input sentences s_1 and s_2 through BERT and get their corresponding d-dimensional pooler output u and v. Then, we follow GLUE(Wang et al. (2018)) to pass the $(3d+1)$-dimensional vector $[u; v; |u-v|; u*v]$ through a MLP to get the final logit. We use sigmoid function to turn the output logit into the probability of being positive and use cross entropy loss to train this head.

Semantic Textual Similarity We separately pass the input sentences s_1 and s_2 through BERT and use a linear layer to project their corresponding pooler outputs to d-dimensional vectors u and v. Then, we use cosine similarity between u and v as the predicted similarity. Since we don't need to predict whether two sentences have opposite meaning, we clamp the predicted similarity to $[0,1]$. We divide the labels by 5 to match the output range of this head and use MSE loss to train it.

4.2 BERT Architecture

Our model inherit from the minBERT backbone architecture while replacing the linear layers in self attention module with LoRA layers and the MLP layers in the transformer blocks with MoE layers.

Multi-Task LoRA with Dissociated Matrices ("lora-no-shared") In LoRA (Hu et al. (2021)), the large attention weight matrices in the pre-trained model are decomposed into two small, low-rank matrices. Given the pre-trained weight matrix $W_0 \in \mathbb{R}^{d*k}$, the weight update during fine-tuning is constrained to be $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d*r}$, $A \in \mathbb{R}^{r*k}$, and rank $r \ll \min(d,k)$. Here, W_0 remains frozen, while A and B are trained to capture the essential features for the model. A scaling hyperparameter α is used. The modified forward pass with input tensor x becomes:

$$h = W_0 x + \frac{\alpha}{r}\Delta W x = W_0 x + \frac{\alpha}{r} BAx$$

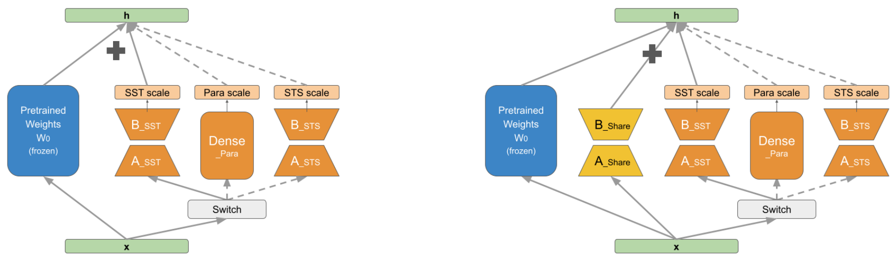
In the multi-task scenario, we proposed a dedicated LoRA matrix, ΔW_i, for each task i, and a task-specific scaling parameter, γ_i. The forward pass in our multi-task model can be expressed as:

$$h = W_0 x + \frac{\alpha}{r} * \gamma_i * \Delta W_i x = W_0 + \frac{\alpha}{r} * (\gamma BA)_i x, \ \ i \in [1, n]$$

Here, the n different tasks' forward passes are routed to their corresponding LoRA matrices and scaling factors. The LoRA matrix A is initialized with Kaiming uniform distribution, B is initialized as zero matrix, and γ is initialized as zero. We set $\alpha = r$. This architecture, denoted as "lora-no-shared," dissociates the LoRA configuration between tasks. We also explore sharing of LoRA matrices between tasks, with details provided in the Experiments section. A visual representation of the multi-task LoRA model attention architecture can be found in Figure 1a. We replaced all query and value modules in the BERT self attention layers with the multi-task LoRA linear layers described above. Finally, to better capture features specific to paraphrase detection, we chose to represent the attention layer for paraphrase with a dense linear layer.

MoE Layer Proposed by Shazeer et al. (2017), a Mixture-of-Experts (MoE) layer consists of a router $h(x)$ and a set of experts $\{E_i(x)\}_{i=1}^N$. The router produces N-class logits for a given input token x and is parameterized by a single weight matrix $W_r \in \mathbb{R}^{N \times d}$. The logits are then normalized via a softmax operation and become a routing probability $p(x)$. Based on $p(x)$, a routing algorithm (e.g. top-k, probability threshold) determine the set of experts $\mathcal{T}$ to route the token, and the output y of the layer is the sum of each expert's output weighted by the corresponding routing probability.

$$y = \sum_{i \in \mathcal{T}} p_i(x) E_i(x)$$



(a) Lora without shared matrices (b) Lora with shared matrices.

Figure 1: Multi-task LoRA model attention module architecture

In this project, we follow the routing algorithm proposed by Zoph et al. (2022). Given $p(x)$ and hyperparameters n and ϵ, we first find out the top-n experts $\mathcal{C}$ with the highest routing probability. We then normalize the top-n routing probability to $\hat{p}(x)_i = \frac{p(x)_i}{\sum_{i \in \mathcal{C}} p(x)_i}$. The token is always routed to the expert with the highest router probability $\arg\max_{i \in \mathcal{C}} \hat{p}(x)_i$. For the other $n-1$ experts, the token is routed with probability $\min(1, \frac{\hat{p}(x)_i}{\epsilon})$.

To integrate MoE layer into a specific transformer layer of BERT, we proposed to initialize all experts with the pre-trained FFN weights. We also set $n = N$ and $\epsilon = 0$, so every token is routed to every expert at initialization. With this design, the output is identical to the pre-trained BERT model, so our model does not lose any knowledge at this step. To further encourage experts to specialize as the fine-tuning goes, we use a linear schedule to increase the gating threshold ϵ, so tokens are gradually routed to the most suitable experts.

To train with MoE layers, an auxiliary load balancing loss is usually used in addition to the original task loss. We use the load balance loss introduced by Fedus et al. (2021). We also follow Zoph et al. (2022) to include the router-z loss to make training more stable. Due to the page limit constraint, we refer readers to the cited paper for detailed information about the losses.

4.3 Multitask Training

To train with multiple tasks simultaneously, each batch of data consists of sub-batch from each dataset, and the model is trained with the following loss function

$$L = L_{\text{sst}} + L_{\text{para}} + L_{\text{sts}} + w_{\text{load\_balance}} \cdot L_{\text{load\_balance}} + w_{\text{router\_z}} \cdot L_{\text{router\_z}}$$

where $w_{\text{load\_balance}}$ and $w_{\text{router\_z}}$ are hyperparameters. We set them to 0.01 and 0.001 respectively for this project.

5 Experiments

We organize this section in the following way. We'll first introduce the tasks of interest and how we evaluate the model's performance on these tasks. Then, we'll present our experiments with the question in mind and use the results to conclude what we learned.

5.1 Tasks and Data

In our experiments, we consider the following three tasks and datasets.

1. Sentiment Classification - Stanford Sentiment Treebank (SST) dataset (Socher et al. (2013)): It includes 11855 single sentences, and each sentence has a label of negative, somewhat negative, neutral, somewhat positive, or positive. For example, the sentence "Light, silly, photographed with colour and depth, and rather a good time" is labeled as positive.
2. Paraphrase Detection - Quora Dataset[1]: It consists of 404,298 question pairs with labels indicating whether particular instances are paraphrases of one another. For example, the

[1]`https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

sentence "I am a Capricorn Sun Cap moon and cap rising...what does that say about me?" is labeled as a paraphrase of the sentence "I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?".

3. Semantic Textual Similarity - SemEval STS Benchmark Dataset (Agirre et al. (2013)): It includes 8,628 different sentence pairs of varying similarity on a scale from 0 (unrelated) to 5 (equivalent meaning). For example, "The bird is bathing in the sink." and "Birdie is washing itself in the water basin" is labeled as completely equivalent while "John went horseback riding at dawn with a whole group of friends." and "Sunrise at dawn is a magnificent view to take in if you wake up early enough for it." is labeled as completely different.

5.2 Evaluation Methods

In our experiments, we consider the following metrics.

1. Sentiment Classification: classification accuracy
2. Paraphrase Detection: classification accuracy
3. Semantic Textual Similarity: Pearson correlation of the true similarity values against the predicted similarity values

5.3 Q1: How does output head parameterization affect performance?

For each task, we experimented with multiple output head parameterizations before getting to the finalized version described in Section 4.1. For each pair of comparison in this section, we fine-tune the same set of backbone parameters and keep all the hyperparameters the same. The only difference is the output head parameterization.

Sentiment Analysis Besides directly using the pooler output from BERT, we also tried to add another attention layer to extract task-relevant information. Specifically, we project the pooler output u to a query and all tokens except for the class token to keys and values. We then concatenate the attention output v with the original pooler output u and feed it to a linear layer for final predictions. The dev set results are shown in Table 1. We can see that although using both u and v gives us a slight improvement when fine-tuning full model, the performance drops when we only fine-tune the LoRA parameters. Since we don't want to fine-tune the whole model for each task, we decided to use only the pooler output.

Input to the output head	Fine-tuned backbone parameters	SST Acc.
pooler output	full model	0.516
pooler output + attention output	full model	0.520
pooler output	LoRA	0.526
pooler output + attention output	LoRA	0.480

Table 1: Performance of different SST output heads

Paraphrase Detection Besides the GLUE head parameterization, we also tried simply concatenating the pooler output of two sentences and feeding it to either a linear layer or a MLP. The dev set results are shown in Table 2. We can see that by providing more information such as element-wise difference or using a deeper output network, we are able to achieve a better performance.

Semantic Textual Similarity Besides the cosine similarity approach, we also tried simply concatenating the pooler output of two sentences and feeding it to a linear layer. The dev set results are shown in Table 3. We can see that cosine similarity provide a huge performance jump because it posses the inductive biaas that two similar sentences should have their embedding vectors pointed towards similar direction.

Output head parameterization	Fine-tuned backbone parameters	Para Acc.
GLUE head	full model	0.889
concat + MLP	full model	0.857
concat + linear	full model	0.809

Table 2: Performance of different Para output heads

Output head parameterization	Fine-tuned backbone parameters	STS Acc.
cosine similarity + clamp	full model	0.667
concat + linear	full model	0.371

Table 3: Performance of different STS output heads

5.4 Q2: What hyperparameters affect LoRA the most?

Building on the original LoRA implementation by Hu et al. (2021), this work explores the impact of two critical parameters: the decomposed matrix rank (denoted by r) and LoRA matrix scaling (denoted by α). We investigate how varying these parameters affects model performance on the Sentiment Classification task. The results are presented in Table 4.

Rank r	Scaling α	SST Acc.
8	8	0.500
4	8	0.503
16	8	0.494
8	4	0.501
8	16	0.365

Table 4: SST accuracy by varying r and α

Our experiments revealed that tuning the LoRA matrix scaling parameter, α, has a more substantial effect on model performance compared to the decomposed matrix rank, r. This finding led us to investigate whether the optimal α scaling for multi-task learning could be learned through training. We compared two approaches: fixing task-specific scaling parameters and fine-tuning them during multi-task training. As shown in Table 5, models with fine-tuned task scaling parameters achieved higher performance. This observation motivated our decision to include a trainable task scaling parameter in our final multi-task LoRA model architecture.

5.5 Q3: Can we get benefit from multitask learning?

Up to this point, we already have a parameter-efficient way to fine-tune BERT for each downstream task. We are curious about whether we can achieve even better performance from multitask learning. However, from our initial experiments, we found out that simply fine-tuning the whole model using the summation of three task losses hurts the performance on all three tasks. This implies that there might be task interference within these tasks. Therefore, we are wondering if we can designate a specific LoRA matrix to store the transferrable knowledge while keeping task-specific knowledge in their own LoRA matrices. Additionally, we explored two-phase approaches to see if separating task-specific knowledge from shared feature training could improve overall performance. Detailed description of the approaches are listed below, and the dev set results are summarized in Table 6.

Shared LoRA matrix (lora-shared) In contrast to the "lora-no-shared" architecture described in Section 4.2, we constructed a "lora-shared" architecture that trains a shared attention LoRA matrix in addition to task-specific matrices (Figure 1b). Both shared and task-specific LoRA matrices are trained simultaneously during a single training pass.

Model	Task scaling	SST Acc.	Para Acc.	STS Corr.
lora-no-shared	Frozen	0.478	0.852	0.779
lora-no-shared	Trainable	0.483	0.864	0.819

Table 5: Comparison between freezing and training task scaling factor

Capturing shared features from task-specific matrices (task-to-shared) This two-phase approach leverages pre-trained knowledge. We first load task-specific LoRA matrices trained with the "lora-no-shared" architecture. These task-specific matrices are then frozen, while we further fine-tune a shared LoRA matrix and the task scaling factors.

Task specialization from shared matrices (shared-to-task) This two-phase approach starts with a general foundation. We first load a shared LoRA matrix trained without any task-specific matrices. Then, we freeze the shared matrix and fine-tune on task-specific LoRA matrices and task scaling parameters.

Model	SST Acc.	Para Acc.	STS Corr.
lora-no-shared	0.483	0.864	0.819
lora-shared	0.473	0.813	0.793
task-to-shared	0.469	0.873	0.821
shared-to-task	0.471	0.632	0.801

Table 6: Performance of multi-task LoRA model with different parameter sharing frameworks

We observed that the LoRA architecture without shared matrices yields the optimal result. Since all the methods we tried so far focus on fine-tuning the self-attention layers, we are then wondering if fine-tuning FFN would be the key to greatly improve performance. Therefore, we experimented with MoE layers, hoping the router can learn to decide what knowledge is transferrable and what knowledge is task-specific. We use lora-no-shared for self attention layer and swap every FFN layer in the pre-trained BERT model following the procedure described in Section 4.2. We experimented with different final values for the gating threshold schedule, where the schedule would increase ϵ from 0 to a final value linearly through 2500 gradient steps, and different number of experts per MoE layer. The dev set results are shown in Table 7. We found out that using MoE does not provide any benefits compared to separately fine-tuning BERT for each task using LoRA, suggesting knowledge sharing is hard to this combination of tasks.

Num experts per MoE layer	Final gating threshold	SST Acc.	Para Acc.	STS Corr.
8	0.2	0.500	0.854	0.760
4	0.2	0.504	0.858	0.730
4	0.4	0.486	0.860	0.738
4	0.6	0.490	0.856	0.736
4	0.0	0.509	0.861	0.724

Table 7: Performance of Adding MoE on top of lora-no-shared

5.6 Results

Through the explorations above, we came to the final model architecture design with output heads for the three tasks respectively being a pooler output + linear for SST, GLUE head for Paraphrase, and cosine similarity with clamp for STS. We replaced the query and value modules in the self-attention backbone with our multi-task LoRA layer with task scaling and without shared LoRA matrix (lora-no-shared). The detailed model architecture can be found in Section 4.2. Our model successfully reduced the training parameters from 111.2M to 16.5M parameters (85.2% reduction).

We trained with batch size 16, 10 epochs and 5000 steps in each epoch. We chose a learning rate of 0.001 for SST and STS modules and 0.00001 for the Paraphrase module.

In comparison with the simple full-model fine-tuning, our model is able to achieve better accuracy with fewer trainable parameters as shown in Table 8.

Model	Trainable parameters #	SST Acc.	Para Acc.	STS Corr.
Full-model fine-tuning	111.2 M	0.486	0.765	0.667
Our model	16.5 M	0.483	0.864	0.819

Table 8: Our model compared to full-model fine-tuning on Dev set

Our model scores on the test dataset SST test accuracy: 0.501, Paraphrase test accuracy: 0.864, STS test correlation: 0.802, and the overall test score: 0.756.

6 Analysis

Initially, we expect a carefully design parameter sharing scheme can help improve the performance. However, from the experiment results shown in Section 5.5, we cannot observe any benefits. To further understand why we cannot improve beyond lora-no-shared, we looked into the training statistics and found out the training set performance of lora-no-shared is almost perfect. This implies the performance problem is not about how to absorb more knowledge from training data but how to close the gap between training set and dev set (shown in Table 9).

Data split	SST Acc.	Para Acc.	STS Corr.
training	0.994	0.921	0.980
dev	0.483	0.864	0.819

Table 9: Performance of lora-no-shared on training set and dev set

7 Conclusion

This work explored strategies for efficient multi-task fine-tuning with large language models. We investigated the impact of task-specific classifier structures and loss functions, demonstrating their significant influence on performance. We then delved into LoRA architectures to reduce trainable parameters. Our findings show that LoRA task scaling plays a critical role, and our novel architecture with disassociated task matrices and scaling factors achieved significant parameter reduction while surpassing full-model fine-tuning performance. Furthermore, we explored parameter sharing strategies for multi-task learning, concluding that a focus solely on sharing LoRA matrices or adapting Mixture-of-Experts architectures was not sufficient. As a future direction, we plan to investigate additional regularization techniques to achieve optimal performance alongside parameter efficiency within the multi-task learning framework.

8 Ethics Statement

Using BERT for paraphrase identification may introduce algorithmic biases due to the composition of the training data. For instance, a model trained primarily on fluent English sentences might unfairly evaluate submissions from non-native English speakers if used to grade tests, as it may fail to recognize their less fluent English as synonymous with correct answers. Additionally, multi-task fine-tuning with shared parameters risks transferring bias from one task to another. On the other hand, implementing an automated system for detecting semantic textual similarity could result in censorship. For example, governing bodies might use textual similarity algorithms to screen and suppress online comments expressing specific political viewpoints. With LoRA significantly reducing fine-tuning costs, it becomes easier to create models with harmful intentions.

To address these challenges, it's crucial to train models on an unbiased corpus that includes data from diverse countries and linguistic backgrounds, thereby mitigating algorithmic biases and ensuring fair treatment across different languages and dialects. Transparency is essential in understanding the potential societal risks associated with the model's downstream tasks to combat censorship issues. Establishing international guidelines for appropriate regularization techniques and refusing inappropriate uses of the model can further safeguard against the misuse of such technologies.

References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. *arXiv preprint arXiv:2101.11038*.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961.

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 *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2023. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269*.

Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *CoRR*, abs/1911.03437.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. 2024. From sparse to soft mixtures of experts.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538.

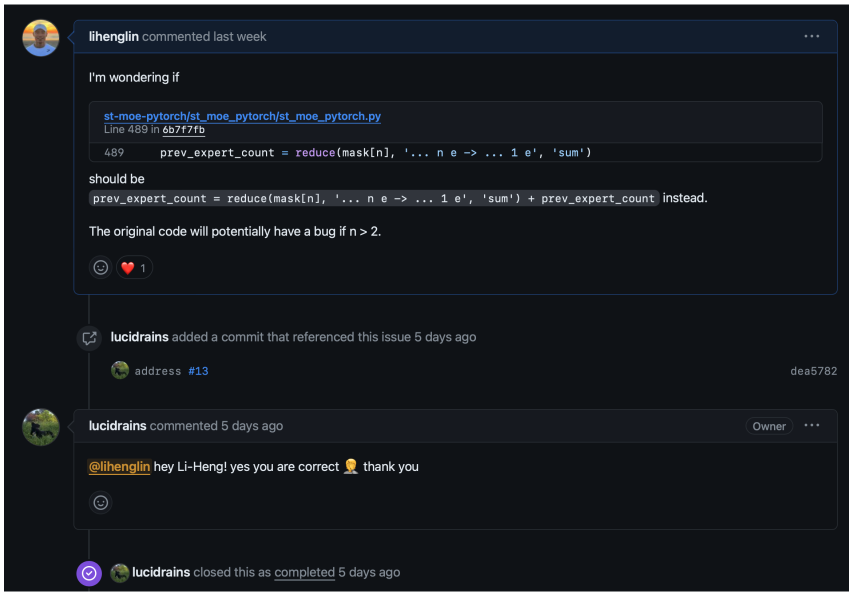
Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

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. *CoRR*, abs/1804.07461.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. 2023. Multilora: Democratizing lora for better multi-task learning. *arXiv preprint arXiv:2311.11501*.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. St-moe: Designing stable and transferable sparse expert models.

A Appendix (optional)



lihenglin commented last week

I'm wondering if

st-moe-pytorch/st_moe_pytorch/st_moe_pytorch.py
Line 489 in 6b7f7fb

```
prev_expert_count = reduce(mask[n], '... n e -> ... 1 e', 'sum')
```

should be
`prev_expert_count = reduce(mask[n], '... n e -> ... 1 e', 'sum') + prev_expert_count` instead.

The original code will potentially have a bug if n > 2.

lucidrains added a commit that referenced this issue 5 days ago

address #13 dea5782

lucidrains commented 5 days ago Owner

@lihenglin hey Li-Heng! yes you are correct 👷 thank you

lucidrains closed this as completed 5 days ago

Figure 2: Catch a bug in the open source ST-MoE implementation we build upon.