Summary of "Low Rank Adaptation and RLHF"

Christopher Jiang (cjiangz), Lingxiao Mou(lmou), Shoma Sawa(shomas)

Problem and Motivation

One common technique in creating NLP applications is to take an existing LLM pre-trained on general domain data and then perform model adaptation to adapt this LLM to a specific domain. One can do this via fine-tuning, which updates all of the parameters in the original pre-trained model. However, as LLMs scale, performing full fine-tuning can become prohibitively expensive due to the number of trainable parameters. Although there have been previous techniques to reduce the number of trainable parameters during model adaptation, they often have higher inference latency, reduced input sequence length, or reduced model quality compared to performing full fine-tuning. LoRA attempts to greatly reduce the number of trainable parameters during model adaptation without additional inference latency or reduced input sequence length while maintaining high model quality by restricting the updates to model weights to low-rank matrices.

However, the original paper of LoRA doesn't focus on scaling, which reduces the overall serving throughput and increases the total latency when serving multiple adapters concurrently. Additionally it did not consider leveraging host memory to increase the number of adapters hosted by a single machine. S-LoRA was later introduced as a scalable LoRA serving system that utilizes batching opportunities in LoRA.

Scaling up a language model does not necessarily make it better at following a user's intent. For example, an LLM might generate outputs that are untruthful, toxic, or not helpful to the user. This is because the objective for many LLMs is to iteratively predict the next token given some input text rather than following the user's instructions helpfully and safely. The goal of the authors of InstructGPT is to align LLMs so that they are helpful, honest, and harmless. The authors try to do this using reinforcement learning from human feedback (RLHF), which fine-tunes an existing LLM using input from humans.

Related Works

LoRA There are other alternatives such as 1) adding adapter layers and 2) prefix training. The original paper by Houlsby (2019) has two adapter layers per Transformer block to impose a similar bottleneck as this paper. This bottleneck adapter adds a small number of parameters to a model by adding the adapter layer to the original network, where the original weight of the network is untouched. Prefix training is an alternative to fine-tuning, where it keeps the model's parameter frozen and optimizes a sequence of “continuous task-specific vectors”. The limitation of this approach is that it is difficult to optimize and performance change non-monotonically in training parameters.

S-LoRA S-LoRA’s memory management was inspired by PagedAttention. PagedAttention is an attention algorithm inspired by virtual memory and paging techniques in operating systems. It can be used to optimize memory for the KV cache.

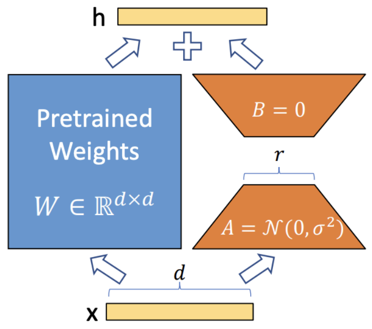
Learning from Human Feedback. 1). RLHF was originally designed for training robots in simulated environments and Atari games and found itself useful in language mode fine-tuning. It uses human feedback to train a reward model and uses the reward model as a function to optimize the reinforcement learning policy. Popular optimization method during this process involves Proximal Policy Optimization.
2). Training language models to follow instructions. The technique seeks a method of cross-task generalization in language models. Specifically, the model is trained with a wide range of datasets and evaluated on a different set of NLP tasks. The difference in training, evaluation, and formatting of instruction is the key consideration in this task.
3.) Mitigating the harms of the language model. Language can produce bias after learning biased inputs. The mitigating methods involve filtering the pre-trained dataset and relying on human agents to perform manual data cleaning.

Solution Overview

LoRA.

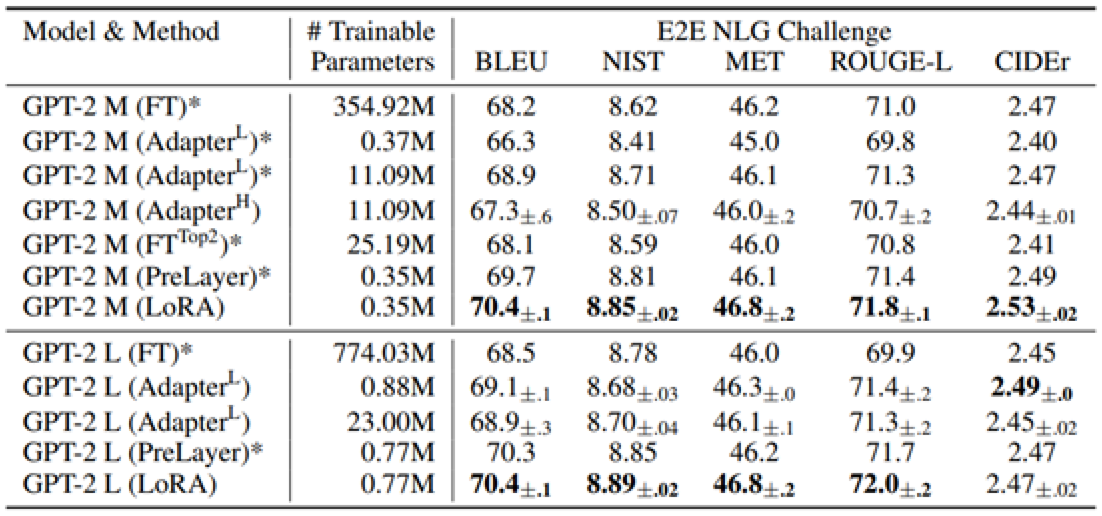
Inspired by previous work indicating that language models have a low “intrinsic dimension” and learn efficiently in subspace, the authors use a low-rank decomposition of weight matrix in language model weights to learn the adaptive information during fine-tuning.

Rather than updating the entire weight matrix during adaptation, LoRA freezes the weights obtained from pre-training and instead learns a low-rank update matrix. Specifically, the final weight after adaptation becomes $W_0 + \Delta W = W_0 + BA$, where W_0 is the original weight matrix and ΔW is the low-rank update matrix with low-rank decomposition $\Delta W = BA$. The matrix A is initialized based on the Gaussian distribution, while B is initialized as the zero matrix. W is also scaled by learning rate to reduce hyperparameter retuning. By applying the rank r approximation, the amount of trainable parameters drastically decreases, reducing the amount of computation needed. By merging the adapter matrix into the original weight matrix, we can avoid the inference overhead of adding an additional module on top of the original weight matrix. Also, BA can be used as a dynamic module to recover W_0 by subtracting BA and then adding another different $B'A'$.


h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Using LoRa reduces the number of trainable parameters. As a result, VRAM usage, and checkpoint size are dramatically reduced, which avoids extensive GPU usage and I/O bottleneck.

The performance of LoRA was evaluated by applying it to popular LLM models, including RoBERTa, DeBERTa, and GPT-2 to WikiSQL, SOTA datasets. For baselines, the authors also implemented a fully fine-tuned LLM, a fine-tuned model that only modifies bias vectors, a fine-tuned model that learns the activations, and a fine-tuned model that does adapter tuning. Overall, LoRA had the best evaluation score. Finally, LoRA was tested on GPT-3, which has a much greater number of parameters. LoRA still outperforms all other models.



Model & Method	# Trainable Parameters	E2E NLG Challenge				
		BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	**70.4**$_{\pm.1}$	**8.85**$_{\pm.02}$	**46.8**$_{\pm.2}$	**71.8**$_{\pm.1}$	**2.53**$_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	**2.49**$_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	**70.4**$_{\pm.1}$	**8.89**$_{\pm.02}$	**46.8**$_{\pm.2}$	**72.0**$_{\pm.2}$	$2.47_{\pm.02}$

Figure 1. LoRA evaluation comparison with baseline models on various evaluation metrics

S-LoRA

S-LoRA supports heterogeneous batching by storing LoRA adapters in main memory and fetching them on demand. The paper uses a custom CUDA kernel optimized for such a task to support batching of multiple adapters with different rank and sequence length. S-LoRA also uses adapter clustering to prioritize batching requests that use the same adapter.
S-LoRA also implements an admission control strategy to drop requests if needed.

For memory management, S-LoRA needs to deal with memory fragmentation from dynamic loading and off loading of adapter weights and the latency overhead from adapter loading and off loading. Unified paging manages both the memory for the KV cache as well as the LoRA adapters, and it can allow for reduced memory fragmentation and increased batch size. S-LoRA also uses prefetching by looking at the current waiting queue and prefetching needed adapters.

The paper uses tensor parallelism for supporting multi-GPU inference for large transformer models, due to its convenience system integration, per-GPU memory and latency reduction. Specifically, the tensor parallelism follows the same strategy as the base model to reduce fusion effort between LoRA and base layer. Intra model parallelism is used for model layers in LoRA followed by an all-reduce operation for the final fused weight matrix. The additional communication cost introduced by LoRA is negligible due to low rank.

InstructGPT

InstructGPT is based on Reinforcement learning from human feedback (RLHF), where the proposed method includes three steps: 1) supervised fine tuning (SFT), 2) reward model (RM) training and 3) reinforcement learning via proximal policy optimization (PPO). Step 1 involves labelers providing desired behavior given a prompt. This is used to train the supervised policy. Step 2 is generating a reward model with the human preference (ranking). An prompt is given to the model, where it will generate several samples. Then the human annotator will rank the outputs given by the model (giving a scalar score). After ranking the outputs, this data is trained to the reward model. The aim of this reward model is to convert a sequence of text to a scalar reward that represents the human preference. Step 3 is where reinforcement learning is used to optimize the original model with the reward function. The supervised policy is fine tuned by using the reward model’s scalar value along with the Proximal Policy Optimization. This approach allows the model to be more aligned with the user.

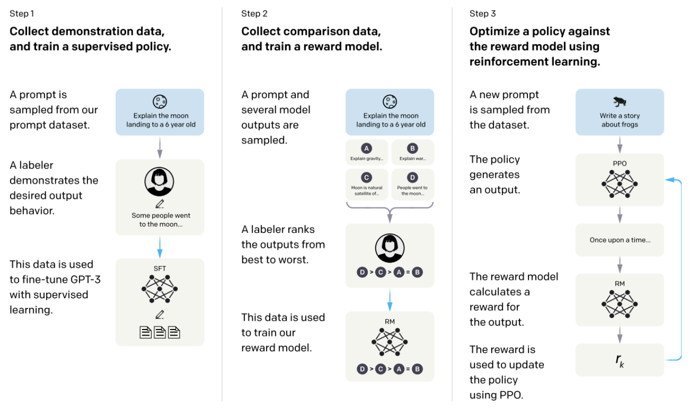

Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

Limitations

InstructGPT: One limitation of this paper is that the human feedback used in this paper was obtained from a small number of contractors that is not representative of the full spectrum of people affected by LLMs, so the feedback may be skewed towards the values of these specific contractors. Another limitation is that the models are still not fully aligned nor fully safe. Furthermore, in most cases, the models follow users' instructions even if it could lead to harm in the real world.

Future Research Directions

LoRA: One future direction is to combine LoRA with other efficient adaptation methods. Another future direction is to explore the mechanism behind fine-tuning/LoRA (how are features learned during pre-training transformed to do well on downstream tasks during adaptation). The authors believe doing so is more tractable for LoRA than full fine-tuning. A third future direction is to come up with more principled ways to decide which weight matrices to apply LoRA to, since the authors use heuristics for this. A fourth future direction is to explore whether the original weight matrix trained during pre-training can also be rank-deficient (not just the adaptation matrix).

S-LoRA: The paper mentions future works such as supporting additional adapter methods, enhanced fused kernels and use of multiple CUDA streams.

InstructGPT: From the results of this paper, the authors draw lessons for future alignment research. The first lesson is that the cost of increasing model alignment is modest relative to that of pretraining, and RLHF is very effective at making language models more helpful to users, which suggests that currently, increasing investments in alignment of existing language models is more cost-effective than training larger models. The second lesson is that InstructGPT generalizes ‘following instructions’ to settings that the authors did not supervise it in, so it might not be necessary to have humans supervise models on every task they perform, which would be prohibitively expensive. The third is that the authors were able to mitigate most of the performance degradations introduced by their fine-tuning process, which is good for making the alignment technique more likely to be adopted.

For future work, the authors noted that making language models better at following user intentions may make them easier to misuse, and so tools other than alignment should be explored for resolving safety issues associated with LLMs. Also, the question of who the models are aligned to is important and need to be explored.

Summary of Class Discussion

LoRA

One point of discussion regarding LoRA was whether using low-rank adaptation matrices might cause the model to converge in the wrong direction. The presenters commented that the original pretrained weights can be used to prevent too much divergence. The presenter also mentioned that the pretrained model actually does not need too high dimension to keep their accuracy. Furthermore, LoRA uses fewer parameters, which could help prevent overfitting. Also, since we are learning a specific thing during adaptation, the model doesn’t need to learn as much compared to pre-training.

Another point of discussion was whether the reduced number of parameters needed for LoRA could help small companies train models with reduced cost. The presenter pointed out that it will reduce the cost to an extent, as it still has to go through the entire LLM. The presenter also mentioned that LoRA is very accessible, since students can implement LoRA with a single line.

S-LoRA

The audience raised concerns about the concept of main memory during batching and the reason for using intra parallelism given the fact the communication overhead could surpass the benefit of computation saving from LoRA. The former topic was addressed by explaining that the main memory is the CPU. The latter is addressed by pointing out potential communication delay using a single GPU. If there is only 1 single GPU responsible for all LoRA adapter result communication, it might cause a lot of waiting for various training weight layer communication.

Training Language Models to Follow instructions with human feedback

One topic that was brought up during discussion was the A/B testing used by ChatGPT. For context, ChatGPT sometimes gives two responses to the user's input and asks the user to choose which one they prefer. A student wondered whether this A/B testing could be used by OpenAI as data for RLHF. The presenters speculated that one reason why OpenAI might do this is to get data about which models users prefer, and another reason might be to train a reward model for RLHF

Another question regarding the A/B testing used by ChatGPT was how to clean the data from the users, since a user might just click a response randomly to get rid of the prompt. The presenters commented that one might be able to use factors such as how long users took to respond to the prompt. It might also be possible to only do A/B testing on users who give good responses.

One question regarding RLHF was why we use a binary comparison scale rather than normalized scalar values for scores. One reason was that people might forget their own scale, leading to bad normalization results.