Efficiently Tuning GPT-2 - A Case Study

Yumou Liu[*] **Zun Huang**[†] **Haoxiang Tang**[‡] **Yingxue Hu**[§] **Su yang**[¶]
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
`{223040100, 223040094, 223040095, 223040087, 223040097}@link.cuhk.edu.cn`

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in general tasks and garnered significant attention in recent years [1, 2, 3, 4, 5]. Various endeavors have been devoted to aligning LLMs with human preferences, including Supervised Fine-Tuning (SFT) [1, 2] and Reinforcement Learning with Human Feedback (RLHF) [6, 7]. These techniques are commonly employed to train and fine-tune LLMs on high-performance clusters and large-scale datasets [5], whose cost is often prohibitive for most researchers. In this work, we use GPT-2 [2] as an example to investigate and propose several techniques to reduce the fine-tuning cost of LLMs.

At a high level, we explore methods to reduce fine-tuning costs, focusing on parameter-efficient adaptation, low-cost preference optimization, and variance reduction in stochastic optimization. In the realm of parameter-efficient adaptation, we investigate the impact of the LoRA [8] rank on model performance. Concerning data-efficient tuning, we analyze how data quality influences the performance of the SFT model. In the domain of low-cost reward modeling, we delve into the DPO algorithm [9], leveraging LLMs directly as the reward model. In the domain of variance reduction in stochastic optimization, we investigate the variance reduction challenge of stochastic gradients under limited batch size. We conduct several experiments on these topics and provide our observations and insights into fine-tuning LLMs.

We further delve into the topics of parameter-efficient fine-tuning and RLHF. In the realm of parameter-efficient fine-tuning, we conducted a comprehensive study on LoRA, exploring how the rank influences the performance of the fine-tuned model. In the context of RLHF, we observed that the DPO algorithm is primarily designed for online reinforcement learning, presenting challenges for many academic researchers who may find it impractical due to the cost associated with online preference data collection. When applying such an online algorithm to offline datasets, there is a risk of introducing high bias, particularly on out-of-sequence (OOS) tokens, as the training data is sampled in an off-policy manner. To address this challenge in offline training, we propose the Conservative DPO (CDPO) algorithm. CDPO introduces a regularizer to the DPO loss function, aiming to constrain the deviation of Language Model predictions on OOS tokens. We conducted several experiments to investigate and validate these aspects.

Besides, we investigate the topic of variance reduction in Stochastic Gradient Descent (SGD) with a small batch size. The technical challenge arises from the dilemma between the need for a large batch size to mitigate gradient variance and the limited GPU memory. Most devices may not accommodate a sufficiently large batch size for tuning LLMs, and gradient accumulation proves time-consuming. In contrast to the conventional approach of accumulating gradients over several batches before taking a step, our proposed method involves updating the model with a small step size during gradient accumulation. The gradients are accumulated in the momentum using running averaging. After several small steps of gradient accumulation, a large step size is applied at once.

[*] Liu proposed the CDPO algorithm and the AGMA optimizer.

[†] Huang wrote code and result analysis for the SFT and the LoRA task.

[‡] Tang participated in the training of the LoRA SFT model and recorded training information.

[§] Hu evaluated SFT versus original GPT-2 models and analyzed performance impact.

[¶] Su wrote code for the LoRA task and provided valuable discussions for the AGMA optimizer.

2 Preliminaries

2.1 RLHF

The RLHF technique has been employed to refine large language models and align them with human preferences [10, 7]. Typically, RLHF consists of two key phases: 1) reward modeling and 2) preference optimization.

Reward Modeling. The primary aim of reward modeling is to train a model capable of discerning human preferences through supervised learning. When presented with the same prompt, an LLM may generate multiple responses with varying hyper-parameter settings. Human experts rank these responses based on their preferences. Utilizing these responses and their corresponding rankings, a reward model is fine-tuned from the pre-trained language model [7] to learn the associated ranking scores.

Preference Optimization. Preference optimization involves fine-tuning LLMs to align them with human preferences by maximizing the responses' ranking scores given by the trained reward model. Typically, this process is modeled as reinforcement learning [10, 7] using Proximal Policy Optimization (PPO) [11], a deep reinforcement learning algorithm. The LLM is denoted as the actor model $\pi_\theta^{RL}(y|x)$ given prompt x and response y with model parameters θ. The parameters of the actor model are initialized the same as the SFT-tuned model, named π^{SFT}. The reward model is denoted by $r_\phi(x, y)$ as the approximated reward function. The most commonly used optimization objective is

$$\max_\theta \mathbb{E}_{(x,y)\sim D_{\pi_\theta^{RL}}} \left[r_\phi(x, y) - \beta \log \left(\frac{\pi_\theta^{RL}(y|x)}{\pi^{SFT}(y|x)} \right) \right], \tag{1}$$

where the first term aims to maximize the reward, and the second term is the KL penalty designed to prevent the policy from deviating from the SFT model, with β controlling the strength of the penalty. Typically, the prompt x is sampled from an offline dataset [5], making this procedure an offline reinforcement learning problem. This procedure incurs significant computational resources as three large models, including the actor, reward, and the SFT model, need to be loaded. Recent work [9] proposed DPO, a method to reduce the computational cost by using the actor directly as the reward model. Specifically, the authors assume the current policy is the optimal policy and derive the formulation of reward functions under the given optimal policy. The optimization target of DPO is to maximize the difference between the derived rewards of the accepted and rejected response pairs. The formulation is

$$\min_\theta \mathcal{L}_{DPO}(\pi_\theta^{RL}; \pi^{SFT}) = -\mathbb{E}_{(x,y_w,y_l)\sim D\pi_\theta^{RL}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta^{RL}(y_w|x)}{\pi^{SFT}(y_w|x)} - \beta \log \frac{\pi_\theta^{RL}(y_l|x)}{\pi^{SFT}(y_l|x)} \right) \right], \tag{2}$$

where $\sigma(\cdot)$ is the sigmoid function, and y_w, y_l are the accepted and rejected responses, respectively.

2.2 Drawback of KL Penalty

Firstly, the KL penalty exclusively focuses on the logits corresponding to the tokens in the sequences x and y while disregarding the logits associated with OOS tokens. For instance, when dealing with the accepted response y_w, the KL penalty aims to push $\pi_\theta^{RL}(y_w|x)$ away from $\pi^{SFT}(y_w|x)$ concerning the tokens in y_w, but it lacks a direct impact on the other logits related to OOS tokens.

Secondly, KL divergence is inadequate for regulating the policy on OOS tokens. To illustrate, consider generating a token with prompt x using three distinct stochastic policies π_a, π_b, and π_c, respectively. If $d_{KL}(\pi_a(y_i|x)||\pi_b(y_i|x)) = d_{KL}(\pi_a(y_i|x)||\pi_c(x))$, it becomes challenging to ascertain whether $\pi_b(x)$ and $\pi_c(x)$ are identical. This is because, if all three policies assign the same probability to the same y_i, the KL penalty, as it stands, does not utilize information from other $y \in \mathcal{Y}$. In our scenario, π_a represents the current policy, π_b is the reference model, and y_i is the next token to be predicted. Consequently, such a KL penalty may not be sufficient to ensure that π_a is similar to π_b, especially on OOS tokens not present in the sequence.

2.3 Variance Reduction in SGD

In this section, we introduce the methods to reduce gradient variance in SGD and the drawbacks of the current methods. The variance reduction techniques in SGD [12] include dynamic sampling, gradient aggregation, and iterate averaging. As for optimizer design, we focus on the last two techniques.

Gradient Aggregation. The gradient aggregation methods reduce variance by reusing previously computed information. Specifically, at time step t, SVRG [13] maintains a copy of the historical parameter θ_k where $k < t$. It computes a batched gradient $G_{\theta_k} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_{\theta_k}(x_i)$ and derives an unbiased estimator of the current gradient by $\mathbb{E}[\nabla R_{\theta_t}] = \nabla f_{\theta_t}(x_t) - (\nabla f_{\theta_k}(x_t) - G_{\theta_k})$, where x_t is a sample from the input space. SAGA [14] stores the historical gradient for each data sample and estimates the current gradient using the average of the historical gradients.

Iterate Averaging. The iterate averaging method [15] stores the parameters after each SGD step and returns the average of the stored parameters. Recent advances [16] employ gradient aggregation and yield $\mathcal{O}(1/t)$ rate of convergence for the averaged iterate sequence.

Drawback of the Current Methods Current variance reduction methods either require large memory space, which is not feasible when tuning the LLMs, or have low sampling efficiency. SAGA needs to store a gradient for each data sample, and the memory cost is proportional to the size of the dataset. The iterate averaging method needs to store all the updated parameters, with memory proportional to the number of steps. SVRG needs to sample a large batch in each step to reduce the gradient variance.

3 The Conservative Direct Preference Optimization Framework

In this section, we introduce the CDPO algorithm designed to regulate the policy over OOS tokens of the trained model to align with a reference model. This approach ensures similarity between policies over OOS tokens, mitigating the common issue of overestimation in offline RL scenarios caused by function approximation errors.

To mitigate the risk of unexpected performance degradation of the offline-trained policy, one approach is to update the policy conservatively. Specifically, we optimize the loss on the offline data samples while preserving the policy on the OOS tokens, similar to the SFT model. Inspired by conservative Q-learning [17], we add a regularizer to the loss function to penalize the deviation of logits on the OOS tokens. The regularizer is formulated as follows:

$$\mathcal{R}(\pi_\theta^{RL}; \pi^{SFT}) = \mathbb{E}_{x\sim D,(y_w,y_l)\sim D\pi_\theta^{RL}} \left[\frac{1}{C-1} \sum_{y_i \in \mathcal{Y}\setminus y_w \cup \mathcal{Y}\setminus y_l} \sigma\left(\beta \log \frac{\pi_\theta^{RL}(y_i|x)}{\pi^{SFT}(y_i|x)}\right)\right], \tag{3}$$

where D is the offline prompt dataset, C is the size of the vocabulary, and $\mathcal{Y}$ is the output space of π_θ^{RL}. Unlike Eq. 2, we do not apply $\log$ after $\sigma(\cdot)$ because the sigmoid function is bounded and facilitates hyperparameter tuning. Combining the regularizer and the DPO loss, we obtain the optimization objective of CDPO:

$$\mathcal{L}_{CDPO}(\pi_\theta^{RL}; \pi^{SFT}) = \mathcal{L}_{DPO}(\pi_\theta^{RL}; \pi^{SFT}) + \gamma \cdot \mathcal{R}(\pi_\theta^{RL}; \pi^{SFT}), \tag{4}$$

where γ controls the strength of the conservative regularizer.

4 Accelerating Noisy SGD with Small Batch

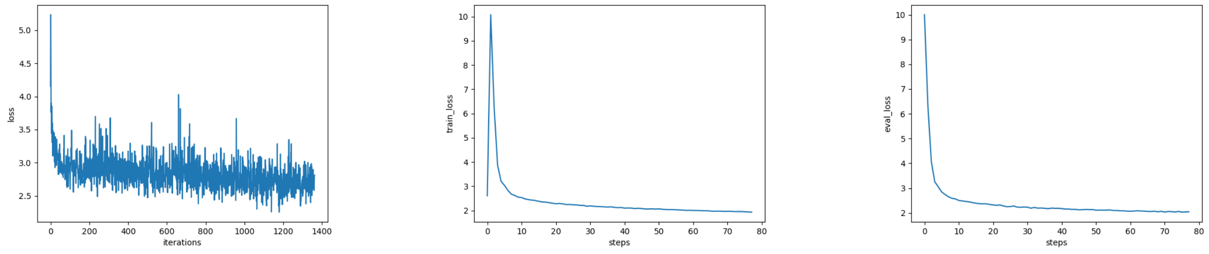
In this section, we introduce the Accumulate Gradient by Momentum Averaging (AGMA) algorithm. The design of AGMA is based on Adam [18]. Our modifications encompass the update procedure of the momentum and the step size strategy. The pseudocode is provided in Appendix A.

4.1 Variance Reduction via Running-Averaging in Momentum

To reduce gradient variance and achieve a consistent update for large steps, we accumulate the gradient of small steps in both the first and second-order momentums. In contrast to Adam and AdamW, where gradients are forgotten with exponentially decaying weights in the momentum, we employ dynamic betas during the small steps. This ensures that the gradients of small steps between two large steps share the same weight as in conventional gradient accumulation.

4.2 Step Size Strategy

The step size strategy governs the relationship between the large step size, denoted as γ, and the small step size. For a given γ, we set the small step size to be $\gamma/\sqrt{K}$, where K is the number of small steps between two large steps. Consequently, the total step size for the K small steps is $\sqrt{K}$, corresponding to K since a larger K implies more gradient accumulation and less variance, providing a reasonable exploration. Additionally, it is unnecessary to design an excessively large step size for small steps, as it could introduce more bias to the accumulated gradient.



(a) Training loss with batch size 4. (b) Training loss with batch size 1024. (c) Evaluation loss with batch size 1024.

Figure 1: SFT training and evaluation loss.

Table 1: Comparison of Average Reward Scores

	Average Reward Score
SFT GPT-2	-3.19
Original GPT-2	-3.37
Test Set Answers	-1.41
Training Set Answers	-0.83

5 Evaluation

5.1 SFT

We conduct experiments using the GPT-2 medium pre-trained model [2] and the HH-RLHF dataset [6]. We employ the Adam optimizer [18] with a learning rate of $lr = 1e - 4$ and $beta = (0.9, 0.99)$. To stabilize gradient descent and conserve limited GPU resources, we utilize the gradient accumulation trick, allowing the batch size to reach 1024, significantly larger than the 64 used in Llama-2 SFT [5]. The performance of the SFT model is evaluated using the auto-regressive loss on both the training and evaluation datasets.

To illustrate the importance of gradient accumulation, we display the training loss under batch sizes of 4 and 1024 in Fig. 1a and Fig. 1b, respectively. The loss curve depicted in Fig. 1a exhibits numerous spikes, indicating significant noise in the gradient. In contrast, the loss curve in Fig. 1b is considerably smoother and decreases more rapidly, suggesting that the larger batch size reduces gradient noise, leading to a faster convergence rate. Fig. 1c illustrates the evaluation loss of the model trained with a batch size of 1024.

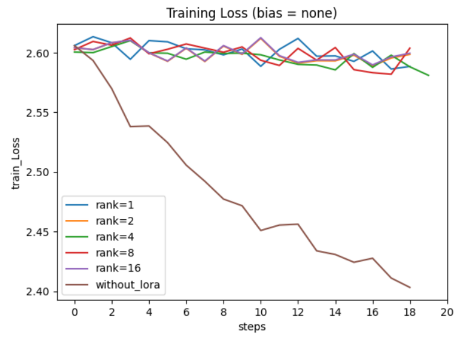
To evaluate the impact of SFT on the GPT-2 model for single-turn dialogue, we compare average reward scores from four scenarios: responses by the SFT model, the original pre-trained model, and actual responses from the test and training sets. Table 1 shows that while the SFT model slightly outperforms the original model on the test set, it still falls short of the naturalness in actual responses from both sets. This highlights the effectiveness of SFT in improving specific aspects of model performance, yet underscores the need for further enhancements to reach the level of real dialogue responses. The discrepancy in scores between the test and training set responses also suggests variations in response quality, with the training set potentially having higher-quality responses.

5.2 LoRA

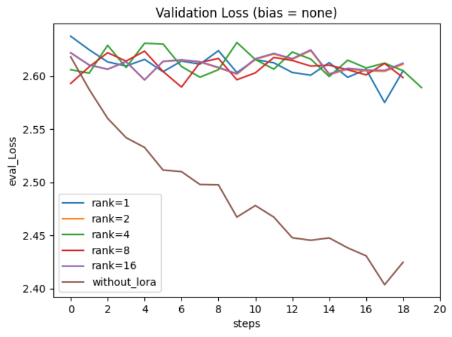
We utilize the GPT-2 medium SFT model for fine-tuning on the SFT dataset. Throughout the experiments in this section, we maintain a fixed batch size of 1 and set the maximum number of steps to 20,000. Various configurations of LoRA were explored, and we analyze the impact of LoRA with experiment results.

The graphs compare the effects of various bias settings and LoRA ranks on training and validation loss. Figure 2 shows the impact of training with and without bias adjustments, while Figure 3 delves into the results of different bias configurations when using LoRA at rank 8.

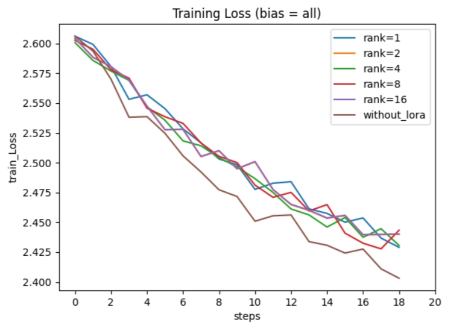
In Figure 2, graphs (a) and (b) illustrate the loss when no bias is trained, with multiple LoRA ranks compared against a baseline with no LoRA. Graphs (c) and (d) show the scenario when all biases are trainable. The graphs show that different ranks of LoRA yield similar loss values and that the performance is close to but slightly inferior to the baseline without LoRA. This suggests that while LoRA allows for a parameter-efficient model adaptation, its advantage in terms of loss reduction is not clear-cut in these specific cases. This also indicates that even at rank=1, LoRA can effectively



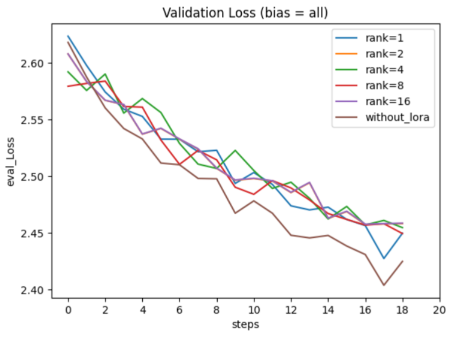
(a) Training loss without training bias.



(b) Evaluation loss without training bias.



(c) Training loss with all bias trainable.



(d) Evaluation loss with all bias trainable.

Figure 2: Training and evaluation loss using different bias options and LoRA ranks.

capture the necessary parameter updates, with performance comparable to the baseline, suggesting that even a minimal rank configuration can work effectively.

Besides, we find that the performance across various LoRA ranks does not demonstrate a consistent pattern, i.e. higher ranks such as 8 or 16 do not consistently outperform lower ranks. This indicates that the choice of rank in LoRA does not necessarily correlate with better or worse performance, highlighting that the effectiveness of LoRA's rank selection may depend on factors beyond just the rank number itself.

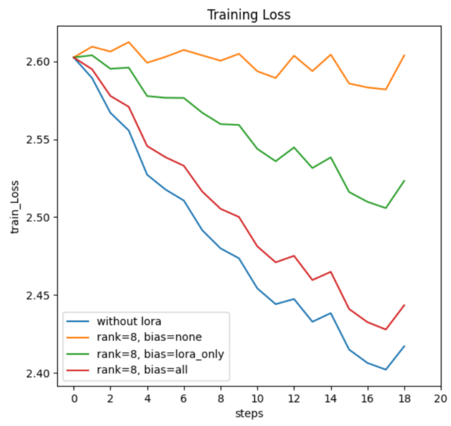
Figure 3 further explores the impact of different bias options when LoRA is fixed at rank 8. Graph (a) shows that training without bias leads to higher loss, whereas enabling bias training, whether partially or fully, yields a noticeable reduction in training loss. Graph (b) for validation loss corroborates this finding, indicating that the presence of trainable biases with LoRA seems to benefit the model's generalization capability.

We also document various aspects of the training process, as depicted in Figure 4. Graphs (a) and (b) illustrate the GPU usage and the percentage of trainable parameters, with and without the use of LoRA. The results demonstrate that even when employing the largest LoRA rank of 16, it still enables a reduction of approximately 1.5GB in GPU memory usage, involving only 1.79% of trainable parameters in the training process. Graph (c) presents the training speed for different configurations. It turns out that incorporating LoRA into the training process leads to a notable speedup of approximately 7.3%.

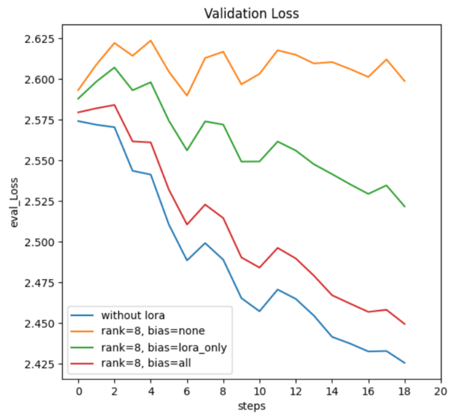
The overall analysis of these results indicates that LoRA is a beneficial technique for adapting large models in a parameter-efficient way. It's crucial to evaluate LoRA's benefits considering the model's complexity and the specific task at hand, as its effectiveness may vary across different scenarios. Furthermore, the flexibility in bias training options offers model developers a tool to balance between computational efficiency and model accuracy. The graphs collectively demonstrate that LoRA can be a valuable component in the fine-tuning process, especially when resources are limited or when working with extremely large models where full model fine-tuning is not feasible.

5.3 DPO & CDPO

We conducted experiments using the GPT-2 medium SFT model obtained in Sec. 5.1 and the HH-RLHF dataset. The AdamW optimizer [19] was employed with a learning rate of $lr = 5e - 6$ and default betas. We utilized gradient accumulation to achieve a batch size of 2048, which is larger than the 512 used in Llama-2 for reward modeling and

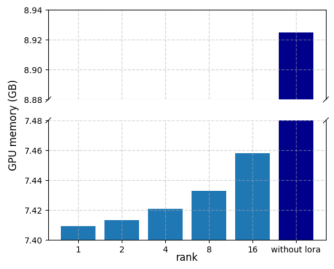


(a) Training loss with different bias options.

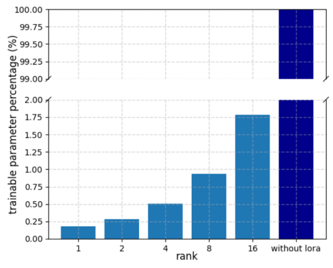


(b) Evaluation loss with different bias options.

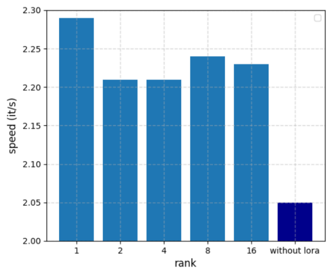
Figure 3: Training and evaluation loss using different bias options with LoRA rank 8.



(a) GPU memory usage (GB).



(b) Percentage of trainable parameters.



(c) Training speed (iterations/second).

Figure 4: Some information during training corresponding to different LoRA ranks.

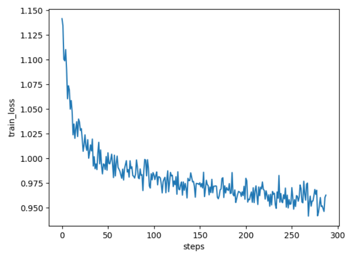
RLHF. The performance of the RLHF models was evaluated using the DPO loss on both the training and evaluation datasets, along with the accuracy of accepted-rejected binary classification on the evaluation dataset. We chose binary classification accuracy as the evaluation metric because the DPO loss is a biased estimator of the LLM performance due to the optimal policy assumption, while accuracy serves as an unbiased estimator.

We plot the training loss, evaluation loss, and classification accuracy of DPO in Fig. 5, respectively. In general, we observe that the DPO algorithm suffers from high variance. This is reasonable from the view of reinforcement learning since DPO does not have a critic for variance reduction. The high variance can be indicated from the spikes in Fig. 5a, and leads to a slow training process. The experiment on each setting takes about 2 days to update around 300 steps and the classification accuracy in Fig. 5c is still low.

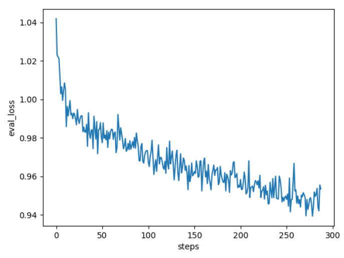
We present the training loss, evaluation loss, and classification accuracy of CDPO in Fig. 6, respectively. It is noteworthy that we use the CDPO loss in Eq. 4 for model training and the DPO loss in Eq. 2 for evaluation. We set $\gamma = 1.0$. In Fig. 6a, we observe that the CDPO algorithm significantly reduces variance, resulting in a higher convergence rate. Comparing Fig. 6c with Fig. 5c, the accuracy of CDPO increases more rapidly than DPO, saving about $1/2$ of the GPU time to achieve the same accuracy. Furthermore, comparing Fig. 6b with Fig. 6c, we find that when the accuracy increases, the evaluation DPO loss does not always decrease, indicating that the DPO loss is a biased estimator.

5.4 AGMA

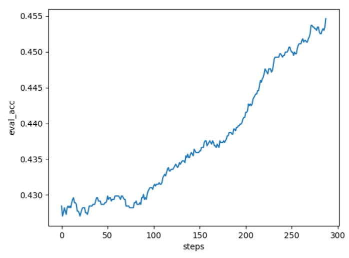
We evaluate AGMA on the DPO task, as DPO poses a high-variance stochastic optimization problem. Figure 7 illustrates the training loss of AGMA and AdamW. The red curve corresponds to the original AGMA, while the orange and blue curves represent other configurations for ablation studies. Each dot of AGMA curves represents the averaged loss of K steps and a large step. A notable improvement in the convergence rate of AGMA compared to AdamW can be observed when comparing the red and blue curves. The orange curve illustrates a scenario where the small step size is set to γ/K, demonstrating the influence of the step size strategy. The green curve represents the update of small steps following the style of SGD instead of AdamW, with γ/K small step size, showcasing the effectiveness of the Adam-style update on



(a) DPO training loss.

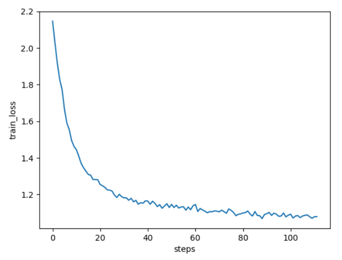


(b) DPO evaluation loss.

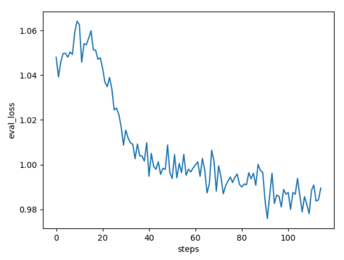


(c) DPO evaluation accuracy.

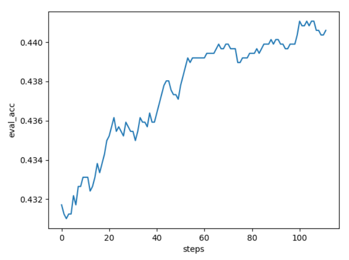
Figure 5: DPO loss and accuracy.



(a) CDPO training loss.



(b) DPO evaluation loss.



(c) CDPO evaluation accuracy.

Figure 6: CDPO loss and accuracy.

the small steps. Besides, we claim that the memory cost of the AGMA optimizer is the same as Adam and AdamW, much lower than SVRG and SAGA.

Beyond Figure 7, we evaluate the performance of different step size strategies. We linearly decrease the total step size for the K small steps and observe that the training loss does not drop after a few large steps. To determine whether the algorithm is stuck in a local minimum or a saddle point, we apply a slight parameter perturbation [20], but the loss does not decrease either. However, this experiment cannot conclusively identify the cause of being stuck, as we have not yet evaluated a stronger perturbation.

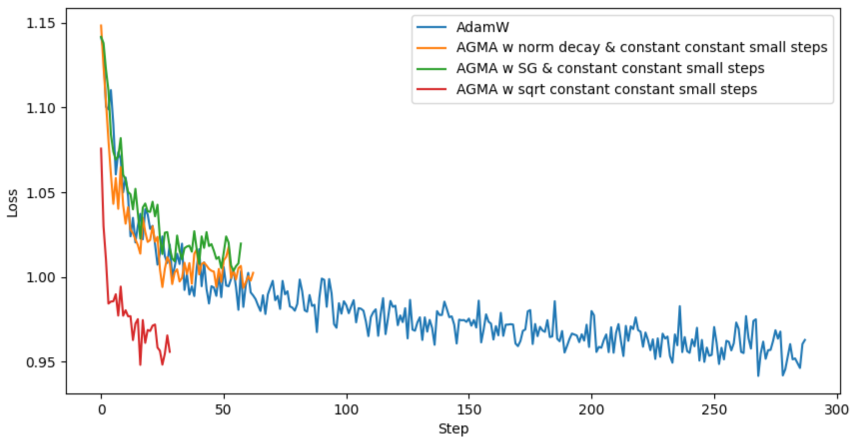


Figure 7: Training loss of DPO with different optimizers.

6 Conclusion

We explored techniques for efficiently fine-tuning LLMs, including SFT, LoRA, DPO, and optimizer design. We identified complexities and introduced the CDPO algorithm. CDPO incorporates a conservative regularizer to expedite the training process. The CDPO algorithm aims to regularize the RLHF fine-tuned LLM, aligning it with the reference model on OOS tokens. This regularization mitigates variance by constraining optimization directions. We proposed the AGMA optimizer, achieving fast optimization with noisy gradients and memory-constrained devices.

Our experiments covered a range of techniques, including SFT, LoRA, DPO, CDPO and AGMA. We observed significant variance during LLM training and mitigated it by employing gradient accumulation. The evaluation of the SFT model incorporated the use of a reward model. Furthermore, we conducted an investigation into the impact of the rank in LoRA on model performance. The evaluation of the CDPO algorithm demonstrated a reduction in variance and a twofold acceleration in training speed, leading to improved accuracy. We evaluated the AGMA optimizer, showcasing a notable improvement in the convergence rate compared to AdamW.

References

[1] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

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

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[5] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[6] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[7] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[9] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

[10] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

[13] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323, 2013.

[14] Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NIPS*, pages 1646–1654, 2014.

[15] Boris Polyak. New method of stochastic approximation type. *Autom. Remote Control*, 7:937–946, 01 1991.

[16] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[17] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[20] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *International conference on machine learning*, pages 1724–1732. PMLR, 2017.

A Accumulate Gradient by Momentum Averaging

```
Algorithm 1: Accumulate Gradient by Momentum Averaging (AGMA)
input  : $\gamma$(lr), $\beta_1, \beta_2$(betas), $\theta_0$(params), $f(\theta)$(objective), $\epsilon$(epsilon), $\lambda$(weight decay), $K$(maximum accumulate
         iterations)
Data: $m_0 \leftarrow 0$, $v_0 \leftarrow 0$
output : $\theta_t$
for $t = 1 \rightarrow \ldots$ do
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$;
    $\tau \leftarrow t\%K$;
    if $\tau = 0$ and $t > 0$ then
        $\gamma_t \leftarrow \gamma$;
        $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t / K$;
        $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 / K$;
    else
        $\gamma_t \leftarrow \gamma / \sqrt{K}$;
        $m_t \leftarrow \frac{\tau}{\tau+1} m_{t-1} + \frac{1-\beta_1}{\tau+1} g_t$;
        $v_t \leftarrow \frac{\tau}{\tau+1} v_{t-1} + \frac{1-\beta_2}{\tau+1} g_t^2$;
    $\hat{v}_t \leftarrow \sqrt{v_t} + \epsilon$;
    $\hat{\theta}_t \leftarrow (1 - \gamma\lambda)\theta_{t-1}$;
    $\theta_t = \hat{\theta}_t - \gamma m_t / \hat{v}_t$;
    if $\tau = 0$ and $t > 0$ then
        $m_t \leftarrow \beta_1 K m_t$;
        $v_t \leftarrow \beta_2 K v_t$;
return $\theta_t$;
```