Exploration into RL-based Language Model Finetuning

Rui Chen
e0659578@stanford.edu
Toronto, ON, Canada

Extended Abstract

Nowadays there have been various RL based methods approaching language model finetuning from different points of view, however, they're mostly trying to solve the same optimization problem despite that their different formulations. This piques our interest into looking for a "unified approach" covering all of them together.

Inspired by this, the report dives into various methods of training LLM's on preference finetuning under a reinforcement learning point of view. It starts with a baseline approach running SFT on smoltalk + DPO on ultrafeedback, then dives into implementing a novel approach AGRO which claims to set up a uniformed framework for most existing methods on RL finetuning. We trained AGRO in both offline and online settings and compared their performance with DPO.

The pipelines are having good performance on a "simpler" preference dataset found online. However, on ultrafeedback the models' ability rate accepted responses higher than rejected never exceeded 53%. The potential guess is preferences are much more complex to learn on ultrafeedback dataset, and further hyperparameter tuning is also required if one hope for further improvements.

Given limited bandwidth and compute resource, we were not able to outperform DPO using AGRO, where in the paper it was claimed to have faster convergence, less variance and better theoretical guarantees. However, their performance are close to each other on submission leaderboard (0.2500 vs. 0.2725), which lies within top percentile among current submissions. We'll extend our effort beyond the end of this project to explore further down this path.

Abstract

In this report, we dived into various RL based methods for finetuning LLM's on learning human preferences. We explored the typical approach of SFT + DPO, then dived into a recent work unifying various existing online/offline methods into a single framework. We tried running preference finetuning on ultrafeedback dataset and got satisfactory results.

Introduction

Fine-tuning large language models, which takes a pretrained base model and aim to improve its performance on various downstream tasks without retraining from scratch, has drawn raising concern in the field of Artificial Intelligence. One popular approach, called "preference finetuning", is to train the model on preference data containing relative feedback. The workflow usually consist of training language model on pairs of (x, y_w, y_l) prompts and accepted/rejected responses, where the goal is to learn rating y_w's over y_l's and model "desired preference" by doing so.

A popular paradigm is treating the problem as a reinforcement learning problem, that is, treating the language model as a policy $\pi(y|x)$, where x is prompt and y is the model's generation. The goal of language fine-tuning turns into training an optimal policy which maximizes the reward of its generation given specific tasks, where reward for preference fine-tuning encourages the model to give higher probabilities to y_w's than y_l's.

Various approaches have been proposed to solve the framework above, main difference lies in training methods (online/offline) and reward modeling (explicit/implicit). Online methods usually train an explicit reward model classifying y_w's from y_l's, then approach to train the policy online maximizing the reward. Offline policies, on the other hand, choose to reparameterize the reward function into policy outputs and training optimal policy with given (x, y_w, y_l)'s only.

This report dives into the offline setting of preference fine-tuning, which trains a Direct Preference Optimization Model warm-started using supervised fine-tuning, then it approaches to implement Any Generation Reward Optimization (AGRO) in both offline and online setting as an extension. I'm going to compare their performance with DPO on both a small dataset found online and ultrafeedback, and analyze the outcome of experiments.

Related Work

The RL-based fine tuning framework can be formulated as maximizing the expected reward following policy π, that is, given some reward $r(x, y)$, a set of policy having form of $\pi(.|x)$, and some mechanism ρ representing support of prompt x, we hope to find an optimal $\pi*$ maximizing the reward with some KL regularization based out of some reference model we start fine-tuning from:

$$G(\pi) = E_{x\sim\rho}[E_{y\sim\pi(.|x)}[r(x,y)] - \beta KL(\pi(.|x)||\pi_{ref}(.|x))] \quad (I)$$

Leave-One-Out REINFORCE (RLOO), proposed by Ahmadian et al. in 2024 (0), visits the problem using a typical online approach. It first trains a Bradley-Terry reward model, which learns to model preference as $P(i \succ j) = \frac{e^{\theta_i}}{e^{\theta_i}+e^{\theta_j}} = \frac{1}{1+e^{-(\theta_i-\theta_j)}}$, and then trains the policy online using typical policy gradient method with leave-one-out baseline, where the gradient can be expressed as:

$$\frac{1}{k}\sum_{i=1}^{k}\left[R(y_{(i)}, x) - \frac{1}{k-1}\sum_{j\neq i} R(y_{(j)}, x)\right]\nabla \log \pi_\theta(y_{(i)} \mid x)$$

Direct Preference Optimization (DPO) , proposed by Rafailov et al. in 2023 (0), dives further into "theoretical roots" of the framework. It proved that when using Bradley-Terry preference model, it can re-parameterize and represent reward implicitly using $\pi(x, y)$ and $\pi_{ref}(x, y)$, resulting in a single training objective in offline setting:

$$-E_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

Recent works in 2024 went even beyond that by deriving the actual analytical form of optimal policy under (I) with any given form of reward:

$$\pi^*(y \mid x) = \frac{\pi_{\text{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right)}{\exp\left(\frac{1}{\beta}V^{\pi^*}(x)\right)}$$

where the normalizing constant is

$$V^{\pi^*}(x) \stackrel{\text{def}}{=} \beta\log\left(\sum_{y\in\mathcal{Y}}\pi_{\text{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right)\right)$$

This inspired the efforts of giving a unified algorithm for solving (I) regardless of training setting and reward function: in early 2025, Tang et al. proposed AGRO algorithm (0), which attempts to maximize generation consistency and approach optimal policy by doing so. We're going to dig further into this algorithm in next section and take as extension of our experiments.

Any Generation Reward Optimization

Given the analytical form of optimal policy, Any Generation Reward Optimization (ARGO) goes beyond above past approaches by trying to solve (I) in general with minimal assumptions. It first proves that the regularized reward $r(x, y) + \beta log\frac{\pi(.|x)}{\pi_{ref}(.|x)}$ is consistent to x under optimal policy, s.t.:

$$r(x,y) - \beta log\frac{\pi_*(.|x)}{\pi_{ref}(.|x)} = some\ \ V(x)$$

Leveraging this finding, the author attempted to optimize (I) by minimizing variance of regularized reward produced by policy. It was proved that it will converge to global optimal in both offline and online settings using loss below, where $\mu(.|x)$ is distribution of y's on the same prompt x in offline setting and is simply $\pi(.|x)$ while applied to online.

$$L(\pi) = \frac{1}{2}E_{x\sim\rho}[Var_{y\sim\mu(.|x)}[r(x,y) - \beta log\frac{\pi_*(.|x)}{\pi_{ref}(.|x)}]]$$

When exploring its behavior in offline settings, the author proved that gradient of this loss compromises the gradient of IPO (implicit preference optimization). Furthermore, he proved that DPO is a special case of ARGO when n = 2 and using Bradley-Terry Model in offline setting for preference fine-tuning. This makes sense as both of them attempted to solve (I) analytically. However, ARGO gives more flexibility by generalizing to any reward function and being capable for both online and offline training.

In online setting on the other hand, the author proved that in online setting the gradient of AGRO is the gradient of RLOO plus another likelihood estimate term:

$$\nabla\mathcal{L}(\pi) = \underbrace{E\left[\nabla f(x,y,\pi)\right]}_{\nabla_{\text{PD}}\mathcal{L}(\pi)} + \underbrace{E\left[f(x,y,\pi)\nabla\log\pi(y \mid x)\right]}_{\nabla_{\text{LR}}\mathcal{L}(\pi)}$$

Training using both the RLOO gradient itself and the ARGO gradient would lead to optimal policy π_*, but the learning dynamic of ARGO ensures less variance & more stabilized training.

Methodologies: Implementation & Setup Details

I am planning to implement 3 workflows of Preference fine-tuning and compare their performance with each other: Direct Preference Optimization, AGRO under offline settings, and AGRO under online settings. The goal is to evaluate how they connect to each other and how their performance differs in preference learning.

For each workflow, I use the Qwen 2.5-0.5B model as a starting point. I first warm-start the model on smoltalk dataset using supervised fine-tuning, then train DPO/AGRO on ultrafeedback dataset to further improve the model's performance. Setups and implementation details on each section are given below.

Warm-start using SFT

Before starting preference fine-tuning, I first "warm-start" our language models on high-quality conversations to improve its capabilities (i.e. fluency & coherence) to an acceptable level so that later policy training converges faster. This is done by training LLM to predict next tokens over carefully selected corpus, where the objective can be expressed as:

$$\max_{\theta} E_{x,y\sim\mathcal{D}}\left[\sum_{t=1}^{|y|}\log\pi_\theta(y_t \mid x, y_{<t})\right]$$

This process is called "supervised fine-tuning" (SFT), where in this stage I use smoltalk dataset to warm up our model. It is a lightweighted dataset helpful for training language models $< 1B$ on instruction following tasks. Due to limited computation resource and GPU memory, I broke each "query-answer" pair into a record to train rather than a round of full conversation, and trim the record to have less than 512 tokens.

To make training lighter and more efficient, I used flash-attention implementation of the base model, and attempt to only train a lora adapter with rank = 16 on top of it. This reduces of trainable parameters by 98.3 % and the same settings will be used in other stages as well.

During training, I split out a fixed validation set of size 200 and closely monitor SFT loss and perplexity/token on both training and validation set to ensure the model is improving.

Preference Fine-tuning using DPO

As mentioned in sections above, it picks up warm-started language model underwent supervised fine-tuning after 12000 gradient steps and set it as reference model. The learned policy is also carried forward from here to optimize DPO loss below, it applies smart reparameterizations so that the problem turned from reward maximization into a supervised classification problem:

$$-E_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

I am training the model with $\beta = 0.1$ on a subset of ultrafeedback datasets, which is a large scale preference dataset created by HuggingFaceF4, consisting of a large number of (prompt, accepted response, rejected response) preference pairs. In order to speed up training and become more memory efficient, I chopped off the data to have at most 128 prompt tokens and 512 response tokens, which make token size to be smaller than 768 for each batch during training.

To ensure the model is learning properly, I am tracking below metrics on both training set and validation set:

- DPO loss
- win rate, specified as $avg(\pi_\theta(y_w|x) > \pi_\theta(y_l|x))$
- kl divergence
- learning rate

However, given the complexity of ultrafeedback dataset, the traced metrics are highly oscillatory and improvements weren't explicit. In order to verify the correctness of my implementation, I ran the same workflow on a small, simple

preference dataset found online to ensure my results were converging over there. I'll present experiment results on the small dataset as well to show the basic correctness of my implementation.

Preference finetuning using offline AGRO

In section 5.1 of AGRO paper, it stated that the gradient of AGRO in offline setting when n = 2 compromise the form of Implicit Preference Optimization (IPO), which is

$$-\beta\left(r_1 - r_2 - \beta\log\left(\frac{\pi(y_1 \mid x)\pi_{\text{ref}}(y_2 \mid x)}{\pi(y_2 \mid x)\pi_{\text{ref}}(y_1 \mid x)}\right)\right)\nabla\log\frac{\pi(y_1 \mid x)}{\pi(y_2 \mid x)}$$

Hence we direct implement the loss form of IPO for simplicity, which gives:

$$\mathcal{L}_{\text{IPO}} = E_{(x,y_w,y_l)}\left[\left(h_\pi(x,y_w,y_l) - \frac{0.5}{\beta}\right)^2\right]$$

where:

$$\begin{aligned} h_\pi(x,y_w,y_l) = &[\log p_\pi(y_w) - \log p_\pi(y_l)] \\ &- [\log p_{\text{ref}}(y_w) - \log p_{\text{ref}}(y_l)] \end{aligned}$$

Similar to DPO, I am also training it on both ultrafeedback dataset and the small preference dataset found online, IPO loss and win rates will be reported on both datasets. However, for training on ultrafeedback dataset, instead of using SFT model as reference model, I'm using the best performant DPO checkpoint from last section as reference model, and see whether IPO can further improve model's performance from there.

Preference fine-tuning using online AGRO

In AGRO paper, the author stated that in online setting the gradient of AGRO is the gradient of RLOO plus another likelihood ratio term:

$$\begin{aligned} \nabla\hat{\mathcal{L}}(\pi) &= \nabla_{\text{PD}}\hat{\mathcal{L}}(\pi) + \nabla_{\text{LR}}\hat{\mathcal{L}}(\pi) \\ &= -\frac{\beta}{n}\sum_{i=1}^{n}\left(R^\pi(x,y_i) - \hat{R}^\pi_{\beta,-i}(x,\pi)\right)\nabla\log\pi(y_i \mid x) \\ &\quad + \frac{1}{2n}\sum_{i=1}^{n}\left(R^\pi(x,y_i) - \hat{R}^\pi_{\beta,-i}(x,\pi)\right)^2\nabla\log\pi(y_i \mid x) \end{aligned}$$

The paper claims the both RLOO gradient and this AGRO gradient with additional LR term guarantees convergence to optimal policy. However, the additional term in AGRO loss provides different learning dynamics leading to faster convergence and better variance reduction.

Given above derivations, when implementing the loss I simply implemented the form below and leave rest of the work to pytorch's autograd funcationalities:

$$\begin{aligned} \hat{\mathcal{L}}(\pi) = &-\frac{\beta}{n}\sum_{i=1}^{n}\left(R^\pi(x,y_i) - \hat{R}^\pi_{\beta,-i}(x,\pi)\right)\log\pi(y_i \mid x) \\ &+ \frac{1}{2n}\sum_{i=1}^{n}\left(R^\pi(x,y_i) - \hat{R}^\pi_{\beta,-i}(x,\pi)\right)^2\log\pi(y_i \mid x) \end{aligned}$$

where $R^\pi(x,y_i)$ is defined as $r(x,y_i) + \beta\log\frac{\pi(y_i|x)}{\pi_{ref}(y_i|x)}$ and detached from backpropagation. To reduce the variance further, I'm taking the paper's advance to subtract a loo baseline from $\left(R^\pi(x,y_i) - \hat{R}^\pi_{\beta,-i}(x,\pi)\right)^2$ as well in $\nabla_{\text{LR}}\hat{\mathcal{L}}(\pi)$

For reward model I was initially considering neumotron-70B but had to give up due to limited inference speed, eventually I picked up a Bradley-Terry reward model available on huggingface, which is trained on ultrafeedback dataset as well using Qwen2.5-1.5B model.

When doing online training, given limited computational resources and infra capabilities (inferencing using vllm/sgLang would be much faster but is memory-intensive), I am training using batch of ONE only - this becomes stochastic gradient descent and would certainly add extensive noise to training. For each prompt x, I am generating 4 responses with max tokens = 300 to speed up training, I am hoping that length of 300 would be enough to give "some signal" for model to start learning on returned rewards.

Performance Metrics

During experiments, we are already collecting certain metrics ensuring that models are learning, where for SFT stage we track perplexity per token and during preference fine-tuning stage we mainly track model's ability to rate accepted responses over rejected ones. In this section, we are discussing more formal ways evaluating model's improvements on learning desired preferences.

For self-evaluation, we are taking a set of 400 prompts from testpref set of ultrafeedback and evaluate our model's performance by looking at their responses. We generate responses w.r.t. each prompt using both fine-tuned models over workflows and base Qwen 2.5-0.5B model before any finetuning, and evaluate the outputs using a Neumotron-70B reward model. The key metric is win rate, which is for how many prompts over the entire test set that our fine-tuned model beats the baseline.

For course evaluation we're following a similar manner as above, the only difference is our fine-tuned model is compared against a stronger SFT model trained by course staff. There's a leaderboard listing the win rate of our submissions, and the passing threshold is 0.1.

Note that to ensure quality of generation and obtain higher win rate, I had to apply various techniques when doing inference. Below comes a list of the major specifications I had to

add to generations, same specifications were added to generations in online training.

- temperature = 0.8, top_p = $0.9 Penalizeonmeaninglessbytetokens$
- Penalize on repetitions
- Enforce vllm engine/huggingface model to stop at end-of-speech and pad tokens

Results

Supervised Fine-tuning

For SFT stage, we observed that both validation loss and validation perplexity were consistently decreasing, indicating that the model was learning and improving well. Furthermore, when submitting onto the milestone leaderboard my trained SFT model obtained a winning rate of 0.48, higher than the threshold of 0.3 on instruction following.

Figure 1: Validation loss over global steps during SFT.

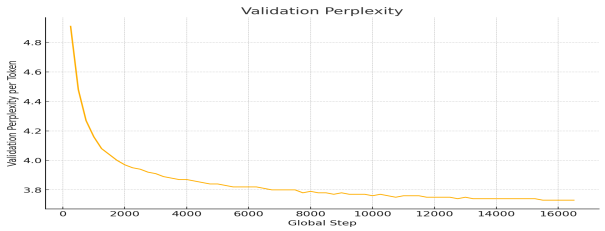

Figure 2: Validation perplexity over global steps during SFT.

Results of offline methods on Small Datasets

For preference fine-tuning, to verify the correctness of my implementations, I ran DPO and offline AGRO pipelines on a toy preference dataset before running on ultrafeedback. This toy dataset consists of only 1100 preference pairs, and I was hoping to validate whether the model would learn in relatively simpler scenarios. Below are the validation losses & win rates I obtained for DPO and offline AGRO:

It is glad to see that both of them are showing fast convergence and win rates are increasing to around 0.9, which is a good sign proving our implementation correctness. We are then running the same pipeline on ultrafeedback to check how much the models can improve over there.

However, it's noteworthy that the validation curve of offline AGRO is much more oscillatory comparing to DPO. I suppose this is because I was using a smaller regularization parameter $\beta = 0.01$ here.

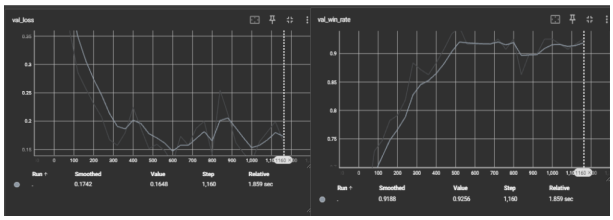

Figure 3: DPO validation loss and win rate

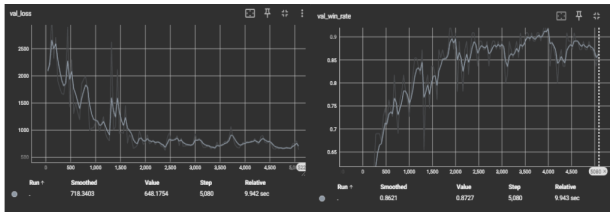

Figure 4: offline AGRO validation loss and win rate

Results of methods on Ultrafeedback

The winrate on ultrafeedback validation set wasn't optimal, for DPO during training the model was able to label at most 53% of accepted responses higher than rejected, where for offline AGRO this number is 49%. The training was highly oscillatory and improvements on validation set is slow, which I'm still working on finding the root causee,

However, the winrate against Qwen-base model was giving good results for DPO, the win rate reached 73% for DPO and 70% for offline AGRO for their most performant checkpoints. While submitting on leaderboard the highest win rate was 27.25%, lying roughly within top 20% percentile among all submissions.

For online AGRO I was only training over 1000 records hence its full potential is not revealed, however in self evaluation I'm already able to get 65% win rate and win rate on leaderboard is 25%. It makes sense that it didn't beat my results on DPO as it wasn't trained on enough data, but it's still descent enough that it's not too far behind.

Discussion - possible missing pointers

To be very honest.... it's unpleasant that I wasn't able to obtain very good results especially on proposed extension. Here comes some of my potential guesses to reasons in such:

For offline AGRO (IPO), it might because I set β too low towards 0.01, which can make training unstable and easy to overfit, hurting its performance.

For online AGRO, it might because my batch size was way too small due to limit of resources. This will definitely bump variance up and hurt its performance, especially given that I was training online version on a 1000 records only.

AGRO might also be more "sensitive" to how the explicit reward is designed, if accepted and rejected responses are having rewards close to each other, then it would be more difficult for AGRO to learn desired pattern. When training in offline setting I was trying with several other reward models before switching the IPO form (i.e. score contained within dataset, neumetron score), non of them had win rate converged even on the toy dataset I was playing around with.

The performance of models are heavily dependent on the hyperparameter setup: batch size, warm up steps, regularization, lora rank..... yet I haven't had bandwidth to explore into the optimal hyperparameter setting for each model to set up a fair comparison.

Conclusion

In this report, we:

- Explored into various RL approaches training LLM on preference finetuning
- Trained a baseline SFT + DPO pipeline, and obtained good performance on win rate against SFT
- Digged into Any Generation Reward Optimization, which unifies offline and online settings with more flexibility and better theoretical guarantees.
- We tried implementing and training AGRO in both offline and online setting and were able to get "okay" results, however, we didn't have time to finetune the model well enough to beat the DPO baseline.

I think the biggest lesson I learned from this project was... it actually requires "deep insights" into the contexts around - see how ARGO derives a uniform framework from analytical form of optimal policy - if one hopes to actually dive into the field of AI/computational extensive methods and make meaningful contributions, rather than throwing the data into fancy models and praying it will work. The paper impressed me a lot from this sense, though it is a pity that I was not able to replicate its theoretical advantages.

Given that.. it also reminds me that it requires extensive engineering efforts to find the optimal experiment settings to achieve expected results, and only then one can say he knows AI and how to use it properly. It is good to realize that I am still a bit far for both requirements, which I will keep improving myself towards them beyond end of the course.

For future explorations, my main concern is to dig into my setup & implementations of AGRO to actually replicate the performance gain stated in AGRO paper. If had more opportunities, I would hope to extend the scope see how RL can help bringing LLM's to be better at reasoning, which differs "AI" from typical pattern recognition/feature extractors.

References

A Ahmadian et al., 2024, Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs

Rafailov et al., 2023, Direct Preference Optimization: Your Language Model is Secretly a Reward Model

Tang et al., 2025, RL-finetuning LLMs from on- and off-policy data with a single algorithm

Contribution

Rui Chen: responsible for data pipeline implementation; baseline implementation of SFT + DPO on preference data; extension (AGRO) implementation; experiments completion and report write up.