WARP: On the Benefits of Weight Averaged Rewarded Policies

Alexander Palanevich

August 5, 2024

Abstract

WARP [RFV+24] is a recent paper which introduces a new alignment strategy by merging policies weights at three distinct stages. The authors claim that their algorithm improves policies quality and alignment, optimizes KL-reward Pareto front of solutions. The goal of the project is to implement the algorithm and to verify whether it performs well with IMDB [MDP+11] dataset.

Github: https://github.com/Sisha0/WARP.

1 Reward model

We use distilbert-base-cased as a reward model. We fine-tune it on the IMDB dataset to classify reviews as positive/negative. The model is trained with batch size 32, AdamW [LH17] optimizer with learning rate $1.41 \cdot 10^{-5}$, linearly decaying to 0. We also use 16-bit precision training and truncate batch sequences to 512 tokens. Training metrics can be seen here. Then each generated text is rewarded in the following way:

1. If the generated sequence is finished, a probability of a positive class is used as a reward.
2. If the generated sequence isn't finished, it's rewarded with -1.

2 SFT model

We use gpt2-imdb as a reference policy. Because fine-tuning this model during RLHF [OWJ+22] stage as is can't be done within computational resources we have, we will use LoRA [HSW+21]. So we need to add LoRA layers first and fine-tune the model for some iterations to later use it as an initial/reference policy during RLHF stage.

We train with the same setup described in 1. We set LoRA rank to 32 and disable dropouts for LoRA layers. Training metrics can be seen here.

3 RLHF

3.1 Dataset preprocessing

We remove all the reviews with length less than 200 characters. Then we tokenize each review and truncate it to the first 5-20 tokens. We do the same processing for both train and test datasets. We also strip the test dataset to the first 100 reviews to evaluate faster.

3.2 Generation strategy

To generate prompts' completions, we generate not more than 128 new tokens using simple sampling without temperature. By default we generate only one completion per data sample for both train and evaluation stages.

3.3 WARP

We use WARP to train a new policy from the SFT initialization. We train for $I = 2$ iterations with $M = 2$ policies. We set the EMA update rate to $\mu = 0.01$, the SLERP rate to $\lambda = 0.5$ and the LITTI rate to $\eta = 0.5$. We also set the KL penalty to $\beta = 0.1$.

Each policy is trained for $T = 100$ iterations with batch size 64, AdamW optimizer with learning rate 10^{-4}, warming up for 20 iterations and then linearly decaying to 0.

3.4 WARP with RLOO

The WARP paper uses the REINFORCE [Wil92] estimator.

$$\mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(.|x)}\left[R(y,x)\nabla_\theta \log \pi_\theta(y|x)\right], \tag{1}$$

where $R(y,x) = r(x,y) - \beta \mathrm{KL}\left(\pi_\theta(.|x)||\pi_{\theta_{\text{anchor}}}(.|x)\right)$. To improve learning, one can reduce the variance of the estimator in (1), while keeping it unbiased, by subtracting a baseline b that has high covariance with the stochastic gradient estimate of (1):

$$\mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(.|x)}\left[(R(y,x) - b)\nabla_\theta \log \pi_\theta(y|x)\right], \tag{2}$$

We choose to use the REINFORCE Leave-One-Out estimator (RLOO) [KvHW19] to generate multiple completions for each prompt and use them as a baseline.

$$\frac{1}{k}\sum_{i=1}^{k}[R(y_{(i)},x) - \frac{1}{k-1}\sum_{j\neq i} R(y_{(j)},x)]\nabla \log \pi_\theta(y_{(i)}|x) \text{ for } y_{(1)},...,y_{(k)} \overset{i.i.d}{\sim} \pi_\theta(.|x). \tag{3}$$

During the experiments, we generate 4 completions for each batch sample. We use the same experiment setup as in 3.3, but decrease the batch size to 16 (greater batch sizes lead to out of memory errors).

4 Experiments

4.1 Environment

We use two environments to experiment:

1. Local environment with GeForce RTX 4060 Ti GPU.
2. Kaggle's environment with two Tesla T4 GPUs. Each policy is trained on a separate GPU in parallel.

4.2 Results

We compare various WARP setups (with or without RLOO and a different number of iterations I) with the base SFT model in terms of KL and reward values in Table 1. As we can see, all variations provide a reward boost over the SFT model. Also, our improved version with RLOO trained for $I = 2$ iterations outperforms base WARP models trained for $I = 2$ and $I = 3$ iterations in terms of KL and reward.

We notice that in the original paper KL/Reward curves mostly go one under another while I grows (i.e. $I = 1$ curve is under $I = 2$, $I = 2$ is under $I = 3$, etc.). This means that training for more iterations leads to increased reward for the same KL value. But in our case (Figure 1) they are somewhat reversed. We still get higher rewards, but the KL values increase noticably as well. Our guess is that tweaking β (KL penalty) or η (moving average rate after each iteration's end) might help to improve the results. Specifically, decreasing η should lead to more SFT-like initialization on each iteration, which should prevent such increasing KL. Increasing β should also lead to lower KL values, as the penalty grows. The maximum number of generated tokens (which is 128 for all the experiments) might influence the trade-off, too. Lower values should lead to lower KL, but this way most of the sequences will not terminate with EOS token, leading to high reward penalty.

Setup	KL	Reward
SFT	0.0	-0.6104
WARP $I = 1$	0.8549	0.2113
WARP $I = 2$	2.4115	0.3725
WARP $I = 3$	5.6479	0.4400
WARP(RLOO) $I = 1$	1.0642	0.0963
WARP(RLOO) $I = 2$	1.8461	0.5128

Table 1: KL/Reward comparison.

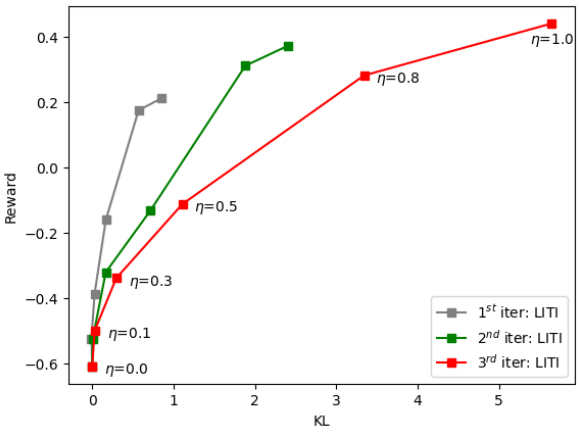

Figure 1: WARP comparison for various I.

The improved RLOO variation (Figure 2) performs better in terms of KL/Reward trade-off, meaning that RLOO's centering strategy might help with bad choices of β, η hyperparameters. Using just 4 generations per sample is enough to see significant behaviour improvements.

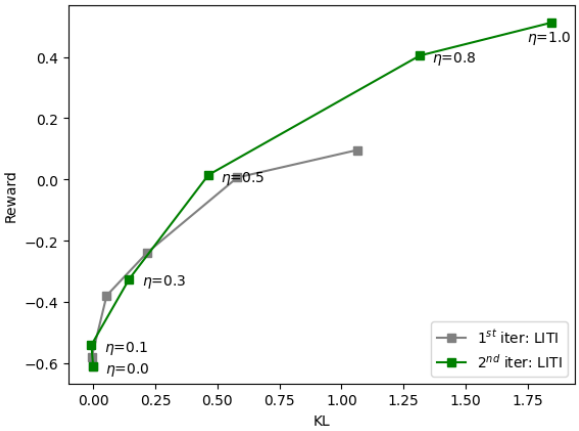

Figure 2: WARP comparison for various I.

References

[HSW+21] 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.

[KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free!, 2019.

[LH17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[MDP+11] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[OWJ+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[RFV+24] Alexandre Ramé, Johan Ferret, Nino Vieillard, Robert Dadashi, Léonard Hussenot, Pierre-Louis Cedoz, Pier Giuseppe Sessa, Sertan Girgin, Arthur Douillard, and Olivier Bachem. Warp: On the benefits of weight averaged rewarded policies. *arXiv preprint arXiv:2406.16768*, 2024.

[Wil92] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.