NoRA: Neuro-Evolution of Low-Rank Adaptation of Language Models

Iheb Gafsi

September 28, 2024

NoRA: Neuro-Evolution of Low-Rank Adaptation of Language Models

Iheb Gafsi * 1 2

[1]INSAT [2]Carthage University

Abstract

Large Language Models (LLMs) such as Llama and Mistral suffer from limited diversity, originality of thoughts on creative writing tasks, convergence to "one point" results, and potentially approaching the desired results from one path when fine-tuning. In this work, we develop an iterative approach to LLM alignment to further elevate the model's capability of novel text generation on downstream tasks. Our method revolves around iteratively creating a population of LoRA adapters, aligning them with IRPO and consequently applying natural selection, customized crossover, and mutation. This effectively resulted an increase of accuracy of 13% on Phi-3-Mini-128K-Instruct and 11% on Mistral-7B-V0.2-Instruct on a story telling dataset. Experiments on small creative writing tasks demonstrated the effectiveness of this method.

1. Introduction

Unsupervised Large Language Models (LLMs) are trained on very large and dense datasets with diverse topics from the open web to encourage their understanding to human natural language, then fine-tuned on a specific task such as question answering, sentiment analysis etc. And eventually aligned with RLHF [Ouyang et al., 2022] or DPO [Rafailov et al., 2023] to maintain the ethics and the specific format of answering. Preference Optimization has proven to outline gains to large language models and align with human preferences compared to supervised fine-tuning (SFT) alone [Ziegler et al., 2019, Stiennon et al., 2020]. Recent works have shown that iterative applications introduce more efficiency and logic to the model making it more informative, whereby elevate the model's precision and hence increase its performance, these methods include Self-Rewarding LLMs [Yuan et al., 2024] Iterative DPO [Xu et al., 2023, Xiong et al., 2024] SPIN [Chen et al., 2024] and Iterative Reasoning Preference Optimization IRPO [Yuanzhe Pang et al., 2024] and other methods [Rosset et al., 2024]. These methods have shown outstanding results in thinking tasks mainly focusing on mathematics and critical reasoning tasks. While other iterative approaches that use generative reasoning models outlined significant performance at reasoning tasks. These methods that revolve around iterative training involving iteration of supervised fine-tuning such as STaR [Zelikman et al., 2022], V-STaR [Hosseini et al., 2024], Quiet-STaR [Zelikman et al., 2024], and Rest^{EM} [Singh et al., 2023] and others have been applied successfully on reasoning tasks.

In this work we argue that the vanilla DPO steers the model towards generating outputs that align with preferred vocabularies, expressions, and concepts.

And we propose a novel method consisting of Neuro-Evolution [O. Stanley, 2002] and IRPO with a Negative Log Likelihood NLL [Yuanzhe Pang et al., 2024]. On each iteration we generate a population of adapters, align them with IRPO+NLL, evaluate them with the corresponding scores, then we apply the principal of "The survival is for the fittest" to finalize the first step. Then we perform a crossover on the LoRA adapters, perform genetic mutations to create adapter mutants and reiterate.

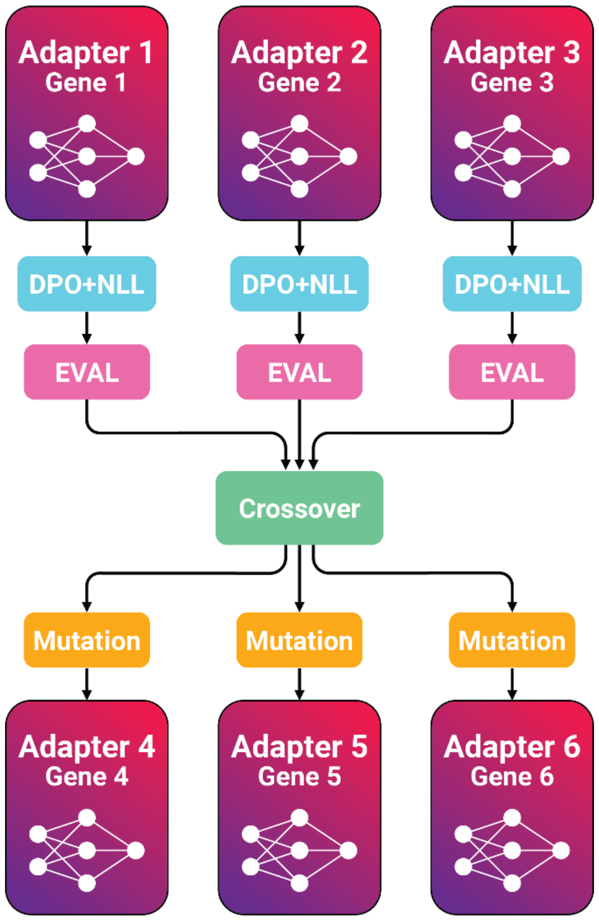

Figure 1: **Overview of NoRA Algorithm.** During training we created a population of 3 adapters and trained them with IRPO, then used natural selection factors, crossed over and then mutants are created and repeat. During inference we only use the fittest among this population that is saved before stagnation.

With this model we achieved an observable increase of performance on a variety of models such as Mistral-7B-v0.2-Instruct and Phi-3-Mini-128K-Instruct where we Phi-3-Mini introduced an increase of 13% and Mistral 11% on a creative writing dataset. Thereby, this experiment demonstrated the effectiveness of this method.

2. Neuro-Evolution of Low-Rank Adaptation:

Population: Initially, create a population of adapters with randomly initialized weights of sizes $W_1 \in \mathbb{R}^{k\times d}$ and $W_2 \in \mathbb{R}^{d\times h}$ corresponding to the original weight $W_m \in \mathbb{R}^{k\times h}$ and d is initialized to 1 at an initial configuration. This specific configuration is crucial for the continuity of the genetic algorithm. The number of genomes in the population isn't scalable for large LLMs (~24B or more). However, it ensures the variety of adapter solutions to obtain. Hence, for smaller models like Mistral-7B and Phi-3-Mini it is advisable to populate your adapters as far as your ressource on your cloud provider such as GCP, AWS, and Azure is able to compute.

The quantity of adapters will open the portal for different approaches to the result which will eventually result an induced potential, whereby the model discovers new patterns for each adapter that will later be combined together in the final outcome.

IRPO + NLL: We first begin with the current adapter A_t, and we generate N different responses for every input x where every response consists of Chain-of-Thought CoT reasoning c followed by a final answer y:

$$(c_i^n, y_i^n) \sim A_t(x_i) \qquad \forall x_i \in D \ and \ n \in [1, N]$$

One then computes the reward $r_i^n = R(y_i^n, y_i)$ based on the correctness of each of these responses. We construct then a set of generated responses augmented

$$G_i = \{c_i^n, y_i^n, r_i^n\}_{n\in[1,N]}$$

In the next step we construct a dataset of response pairs D_t^{pairs} based on the generations G_i [Yuanzhe Pang et al., 2024]

Given the preference pairs, we can now train a new model A_θ that will becomes the next model M_{t+1}. The parameters θ are initialized from model A_t [Yuanzhe Pang et al., 2024]. The loss corresponding to each preference pair is as follows:

$$\mathcal{L}_{DPO+NLL} = -\frac{\log M_\theta(x_i, c_i^\omega, y_i^\omega)}{|x_i| + |c_i^\omega| + |y_i^\omega|} - \alpha \log \sigma\left(\beta \frac{\log M_\theta(c_i^\omega, y_i^\omega|x_i)}{\log M_t(c_i^\omega, y_i^\omega|x_i)} - \beta \frac{\log M_\theta(c_i^l, y_i^l|x_i)}{\log M_t(c_i^l, y_i^l|x_i)}\right)$$

Natural Selection: On the evaluation stage, adapters would have different genes encoded in their weights and topologies. An evaluation is done on a downstream task for all of the genomes in the population. Whereby, some genotypes will stand out with more features for survival and ones will obtain high scores (reversed fitness scores) and potentially decrease their chances of continuation. The fitness function [O. Stanley et al., 2002] will be the loss where the goal is:

$$\min \mathbb{E}[\mathcal{L}_{DPO+NLL}]$$

The selection will not be limited on genomes with the lowest reversed fitness scores but instead will be used for hidden features.

Crossover: To finalize the first iteration of the NoRA algorithm, all the survived Genomes will be crossed over in pairs, potentially procreating and resulting a new population of their offsprings.

The repopulation process is what gives NoRA effectiveness. We take one pair of adapters A_n and A_m and we combine their genetic codes. For instance, we take the two matrices for each weight of the corresponding adapter: $W_{n1} \in \mathbb{R}^{k\times d_n}, W_{n2} \in \mathbb{R}^{d_n\times h}$ and $W_{m1} \in \mathbb{R}^{k\times d_m}, W_{m2} \in \mathbb{R}^{d_m\times h}$ the corresponding adapter matrices respectively to A_n and A_m, d_n, d_m are the intrinsic dimensions of these weights. In our work we introduced 2 more coefficient matrices with values $\in]0,1[$, we denote these matrices $C_{n1} \in \mathbb{R}^{k\times d_n}, C_{n2} \in C^{d_n\times h}$ and $C_{m1} \in \mathbb{R}^{k\times d_m}, C_{m2} \in \mathbb{R}^{d_m\times h}$ the values of which will be determined with its performance because survival is for the fittest. We then calculate the scaling factors σ, γ to benefit the fittest.

$$V = Softmax\left(\begin{pmatrix}\varphi(A_n)\\ \varphi(A_m)\end{pmatrix}\right) \qquad \sigma = V_1 \quad \text{and} \quad \gamma = V_2$$

The resulting adapter matrices will be the projection of the matrices with higher intrinsic weights on the one with lower one so their sizes would be $W_{R1} \in \mathbb{R}^{k\times \max(d_n,d_m)}, W_{R2} \in \mathbb{R}^{\max(d_n,d_m)\times h}$, as well as summing them so the describing equation would be:

$$W_{R1} = \sigma C_{n1} \odot W_{n1} \oplus \sigma C_{m1} \odot W_{m1}$$
$$W_{R2} = \sigma C_{n2} \odot W_{n2} \oplus \sigma C_{m2} \odot W_{m2}$$

Here $\odot$ presents the Hadamard product and $\oplus$ is the operator for projection and summation.

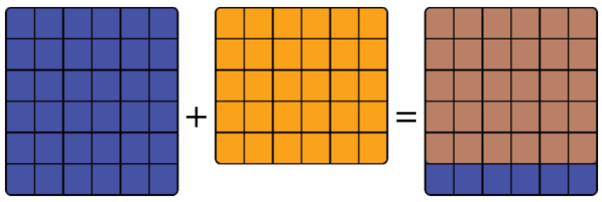

Figure 2: A matrix summation and projection where the first is larger than the second one. The resulting

matrix is the fusion of them, creating an offspring adapter matrix.

Favorizing the genes of the fittest is a crucial part of the evolution of the adapters and hence increases the performance of the model.

Mutation: To finalize the reproduction process, we used different types of mutations such as value mutations and topology mutations. To summarize the model has a certain probability of changing one of its weights, completely or partially, as well as having a probability for adding or deleting an intrinsic dimensionality. Adding a dimensionality works by concatenating a null vector to the extremity of each matrix, and removing one works in the same order. This ensures the innovation of the model and thus increases its chances of obtaining different approaches to final goal.

This method can look similar to the one used by O. Stanley et al. [2002] but is simpler and adapted to a more complex problem such as this one. There are four main differences. First, in the vanilla NEAT we use dynamic neural networks that cannot be manipulated with matrices but instead with graphs, while we use tensor-based neural networks due to the complexity of the model's architecture. Second, NEAT used initial environment fitness functions to determine the model's overall architecture, here we use IRPO's loss as a reverse fitness function to minimize the cost and potentially choose the right adapter. Third, our method introduces coefficient crossover to potentially favor the genes of the fittest. Fourth, due to the different topologies of the adapters (inequivalent intrinsic dimensions) we introduced the projection and addition mechanism.

Table1: **CW6K results** comparing Neuro-Evolution of Low-Rank Adaptation of Language Models (NoRA) against other model baselines trained on the same data and the same iterations. We report exact accuracy match between the two parties.

Model	Test Accuracy (%)
NoRA *(initialized from Mistral 7B v0.2 instruct)*	
Iteration 1	55.1
Iteration 2	56.3
Iteration 3	56.9
Iteration 10	65.7
NoRA *(initialized from Phi-3-mini instruct)*	
Iteration 1	51.6
Iteration 2	55.9
Iteration 3	56.3
Iteration 10	62.8
Other Mistral 7B initialized methods	51.0
IRPO *initialized Mistral 7B v0.2 instruct*	47.2
Zero-shot CoT	

Using NoRA allows the model to surpass the chain of thoughts as shown in Figure 3. This demonstrates the effectiveness of NoRA over small models.

3. Experiments

3.1. Creative Writing CW6K

Our experiments aimed to improve model performance on solving creative writing problems. We utilized the CW6K dataset containing real problems with questions (x_i), human solutions (c_i with answer y_i), and a training set of roughly 7.5k problems.

We began with the Mistral-v0.2 7B model as our seed model. We employed zero-shot prompts (detailed in Appendix A.1) that combined the problem question with instructions to generate solutions in a specific format, allowing for easy answer extraction.

In each of ten iterations, we generated 12 solutions (N) per problem using sampling. The temperature for sampling was set at 0.7 for the first two iterations and 1.3 for the final two, aiming to introduce a higher chance of incorrect solutions later on. Since some problems might not have a model-generated correct solution, the gold human solution (c_i, y_i) was included in the winning set (G_i^w) to ensure it wasn't empty.

Next, we generated 10 solution pairs (K) per problem for training with the loss function described in Equation 1. Pairs exceeding the context length or lacking any incorrect generations were filtered out. This resulted in roughly 6k training pairs per iteration.

For each iteration, we trained the model for a maximum of 2000 steps. We then selected the best checkpoint based on a held-out set of 100 training samples. Finally, we retried the training process including those 100 samples for the chosen number of steps. The training utilized a batch size of 16 and a learning rate of 7e-7. The overall results, including exact match accuracy on the CW6K test set, are presented in Table 1.

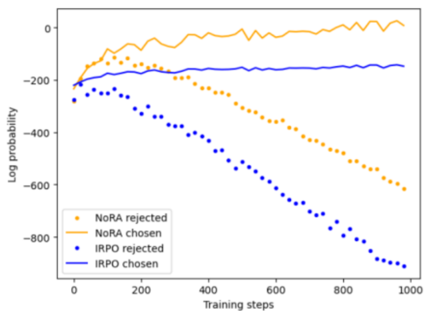

Figure 4: **Effect of NoRA in term of IRPO training.** In our CW6K experiment we kept track on the IRPO loss evolution in comparison to NoRA, the IRPO loss converges much faster than NoRA's. This reveals the effectiveness of NoRA on the SFT trained Mistral.

4. Related Work

Iterative Reasoning Preference Optimization Previous research has shown that iterative preference optimization methods excel at general instruction tuning tasks for large language models (LLMs) [Yuan et al., 2024; Chen et al., 2024]. However, these methods often yield minimal improvement or even hinder performance on reasoning tasks. This work by Pang et al. (2024) proposes a novel iterative approach that optimizes preference between competing Chain-of-Thought (CoT) candidates. Their method focuses on identifying winning reasoning steps that lead to the correct answer, contrasting with losing steps. Training involves a modified DPO loss function incorporating a negative log-likelihood (NLL) term, which they demonstrate as crucial for improvement. Their iterative approach using only training set examples significantly boosts reasoning accuracy on various benchmarks. Notably, they achieve an accuracy increase from 55.6% to 81.6% on GSM8K using Llama-2-70B-Chat, surpassing other Llama-2-based methods that rely on additional datasets.

Methods Improving Reasoning Ability Several recent approaches focus on improving training data for reasoning tasks ([Yu et al., 2023, Toshniwal et al., 2024]). This work, however, takes a different approach by investigating learning algorithms. Existing methods like Expert Iteration, STaR, and ReSTEM all rely on filtering high-quality training examples through various loops and refinements. These approaches differ from ours in that they utilize techniques like rejection sampling and ground truth verification, rather than our focus on pairwise preference optimization. V-STaR is similar in its use of a verifier, but it trains the verifier with DPO, while we train the generator itself. MAPO also leverages DPO, but for a different purpose - achieving multilingual reasoning through translation.

Acknowledgment

We thank our colleagues at the National Institute of Applied Sciences and Technology of Tunis INSAT and the CU for the valuable discussion, especially my professors for their assistance.

References

[1] Kenneth O. Stanley, Risto Miikkulainen. *Evolving Neural Networks through Augmenting Topologies. URL* http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

[2] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint arXiv:2305.18290v2, 2024.*

[3] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie- Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, William El Sayed. Mistral 7B Technical Report. *arXiv preprint arXiv:2310.06825v1, 2023*

[4] Long Ouyang, Jeff Wu, Pamela Mishkin, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Amanda Askell, Jan Leike, Fraser Kelton, Peter Welinder, Luke Miller, Maddie Simens, Paul Christiano, Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155v1, 2022*

[5] 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.*

[6] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural*

Information Processing Systems, 33:3008–3021, 2020.

[7] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. *Self-rewarding language models. arXiv preprint arXiv:2401.10020, 2024.*

[8] Jing Xu, Andrew Lee, Sainbayar Sukhbaatar, and Jason Weston. Some things are more cringe than others: *Preference optimization with the pairwise cringe loss. arXiv preprint arXiv:2312.16682, 2023.*

[9] Wei Xiong, Hanze Dong, Chenlu Ye, Han Zhong, Nan Jiang, and Tong Zhang. *Gibbs sampling from human feedback: A provable kl-constrained framework for rlhf. arXiv preprint arXiv:2312.11456, 2023.*

[10] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. *Self-play fine-tuning converts weak language models to strong language models. arXiv preprint arXiv:2401.01335, 2024.*

[11] Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, Jason Weston, *Iterative Reasoning Preference Optimization. arXiv preprint arXiv: 2404.19733v1, 2024*

[12] Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacroce, Ahmed Awadallah, and Tengyang Xie. Direct nash optimization: Teaching language models to self-improve with general preferences. *arXiv preprint arXiv:2404.03715, 2024.*

[13] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems, 35 :15476–15488, 2022.*

[14] Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457, 2024.*

[15] Eric Zelikman, Nick Haber, Georges Harik, Yijia Shao, Noah D.Goodman, Abstract Varuna Jayasiri. Quiet-STaR: language models can teach themselves to think before speaking *arXiv preprint arXiv:2403.09629v1,2024*

[16] Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, et al. Beyond human data: Scaling self-training for problem-solving

with language models. *arXiv preprint arXiv:2312.06585, 2023.*

[17] Edward Hu, Yuanzhi Li, Yelong Shen, Shean Wang, Phillip Wallis, Lu Wang, Zeyuan, Allen-Zhu, Weizhu Chen. LoRA: low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685v2, 2021.*

[18] Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Git man. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *arXiv preprint arXiv:2402.10176, 2024.*

[19] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284, 2023.*

A. More Details on Experimental Setup

A.1 Prompts

CWK16: Foreach CW6K question, we use the following prompt as the input to the language model:
You are a creative writer, your task is to write an article, going through an organized logical and lexical process to eventually write the corresponding article.
Question: [question]
Answer: