ORPO: Monolithic Preference Optimization without Reference Model: Review

Links:

- Paper: ArXiv Link
- Code: Repo
- HuggingFace Transformers implementation docs
- HuggingFace Transformers implementation source

Alternative methods:

- Reinforcement Learning with Human Feedback (RLHF) using Proximal Policy Optimization (PPO)
- Direct Preference Optimization (DPO)
- Kahneman-Tversky Optimization (KTO)
- Contrastive Preference Optimization (CPO)

Background:

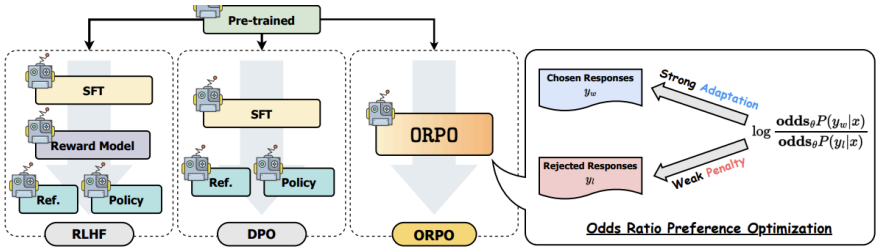

Figure 2: Comparison of model alignment techniques. ORPO aligns the language model *without a reference model* in a single-step manner by assigning a weak penalty to the rejected responses and a strong adaptation signal to the chosen responses with a simple log odds ratio term appended to the negative log-likelihood loss.

Modern LLM training usually comprises 3 main stages:

1. **Unsupervised pre-training** on a large and diverse text corpus
2. **Specialization** using Supervised Fine-Tuning(SFT) for a target domain adaptation. This allows the model to be tailored for our desired use case (e.g. dialogue, following instructions, translation, summarization, etc.) or any other specialized domain we prepare the model for. By fine tuning on domain-specific data and tasks during the training process, the model becomes optimized to excel at the intended application or subject area. This step is supervised and is performed on a labeled dataset (i.e. a set of inputs and expected outputs which is prepared in advance).

3. **Preference Alignment**: A third optimization step incorporates domain-specific preference answer pairs, representing better (“preferred”) or worse (“rejected”) responses to a set of prompts. Both answers are within the domain, but we want to nudge the model towards the preferred answer and away from the rejected one. This allows us to express subtle preferences of how the model should answer which we cannot otherwise encode in previous steps. For each question, there should be a set of answers, and each answer should be labeled with a preference value. The preference values can be binary (e.g., "good" or "bad") or ordinal (e.g., "1" to "5").

In an analogy, we can think of a hypothetical process of making a metal sword. In the first phase, we create a metal block. Then we sharpen it to form the blade - at this point it’s a useful tool for its domain. Finally, we “dull” some parts of the blade to create a handle - incorporating human preference for the usability of the sword.

The third step is usually done using RLHF with PPO (proximal policy optimization) or the newer DPO technique. In this context PPO is an iterative algorithm that updates a model's weights by finding a direction in which the language model is more likely to generate responses that yield high rewards. PPO uses a technique called "trust region" to ensure that the new model is not too different from the old model, which helps to stabilize the learning process and avoid “reward hacking”.

PPO is a two step process: first, a reward model is trained on the question and pairs of answers with a preference relation between them, such that it learns to predict the preference. Second, a reinforcement learning procedure is performed on the model after SFT with the reward model guiding the process. The main issue with PPO is that its training is costly and unstable(e.g. DPO paper, section 5.2).

Instead, DPO (Direct Preference Optimization) was proposed as an alternative which does not require reinforcement learning. DPO aims for the same goal as PPO, focusing on maximizing rewards while adhering to a constraint related to KL-divergence of the fine-tuned and the original model (to avoid “reward hacking”). DPO works by boosting the log likelihood of preferred outcomes over less desired ones and introduces a dynamic weight for each example to avoid issues seen with simpler approaches. DPO uses a theoretical framework (the Bradley-Terry model) to ensure the reward function properly reflects actual preferences.

Note that both PPO and DPO still require a separate step after supervised fine-tuning. The proposed method (ORPO) suggests a solution which aims to remove the need for a third step. Instead the preference pairs are used during the SFT process via an additional term in the loss function.

Following our analogy the incorporation of preferences into an earlier stage is akin to sharpening the sword less where the handle would be, instead of requiring a separate dulling phase.

Proposed Method:

As mentioned ORPO incorporates an additional term to the loss function used in the SFT stage. The challenge is how to incorporate an additional item to the loss without negatively affecting the domain adaptation (the main goal of the SFT process in the second stage). One part of the solution is to scale the added term in the loss function. Another is to make the additional term such that it would encourage the preferred answer but still discourage the rejected one.

This is an essential problem with SFT - the rejected answer is also a legitimate answer from the point of view of domain adaptation. For example, if the goal of SFT is to train our base model to follow instructions, then a rejected answer might still be following instructions (within the domain) even if it's not preferred over the accepted answer. In fact it is shown that by default, the probability of the rejected answer grows during SFT together with the probability of the accepted answer (see Figure 3 in the article). Therefore we don't want to penalize the answer in itself if it's within the target domain, but only in relation to a better answer. How can this be achieved?

The solution uses a loss term that is a ratio between the estimated probability of the less desirable ("rejected") answer and the probability of the preferred ("accepted") answer. Minimizing this ratio-based loss encourages the model to generate outputs that prioritize the most suitable responses.

$$\log P_\theta(y|x) = \frac{1}{m}\sum_{t=1}^{m} \log P_\theta(y_t|x, y_{<t}) \quad (3)$$

$$\mathbf{odds}_\theta(y|x) = \frac{P_\theta(y|x)}{1 - P_\theta(y|x)} \quad (4)$$

$$\mathbf{OR}_\theta(y_w, y_l) = \frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)} \quad (5)$$

$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)}\left[\mathcal{L}_{SFT} + \lambda \cdot \mathcal{L}_{OR}\right] \quad (6)$$

$$\mathcal{L}_{OR} = -\log \sigma\left(\log \frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)}\right) \quad (7)$$

Figure 1

Computation

The loss is computed for each token position ***t*** as follows:

1. For each answer of (preferred, rejected):

a. Run a forward pass with the input followed by all tokens 1...t-1 filled from the answer ("teacher forcing"), to predict the next token. Thus the next token is conditioned on the input and previous tokens of the response: $p(y_t | x, y_{<t})$.
b. Use the result to calculate Eq. (3) up to the current token (in practice this means just adding a single term to the addition and incrementing **m**). We can also immediately compute the value for Eq. (4).

2. Calculate the ratios of Eq. (5), plug it into Eq. (7)
3. For each answer of (preferred, rejected):
    a. Plug the resulting term into Eq. (6) which is the final loss value for the token generated for that answer.

This results in two forward passes, instead of 4 passes required in DPO and PPO (see subsection 7.3 of the article).

Model Name	Size	$\text{AlpacaEval}_{1.0}$	$\text{AlpacaEval}_{2.0}$
Phi-2 + SFT	2.7B	48.37% (1.77)	0.11% (0.06)
Phi-2 + SFT + DPO	2.7B	50.63% (1.77)	0.78% (0.22)
Phi-2 + ORPO *(Ours)*	2.7B	**71.80% (1.59)**	**6.35% (0.74)**
Llama-2 Chat *	7B	71.34% (1.59)	4.96% (0.67)
Llama-2 Chat *	13B	81.09% (1.38)	7.70% (0.83)
Llama-2 + ORPO *(Ours)*	7B	**81.26% (1.37)**	**9.44% (0.85)**
Zephyr (α) *	7B	85.76% (1.23)	8.35% (0.87)
Zephyr (β) *	7B	90.60% (1.03)	10.99% (0.96)
Mistral-ORPO-α *(Ours)*	7B	87.92% (1.14)	11.33% (0.97)
Mistral-ORPO-β *(Ours)*	7B	**91.41% (1.15)**	**12.20% (0.98)**

Table 1: Table of instruction-following abilities of each checkpoint measured through AlpacaEval. While clearly showing the improvements in instruction-following abilities after training with ORPO, it is notable that ORPO models exceed RLHF or DPO models of Llama-2 and Mistral (* indicates the results from the official leaderboard.)

Results:

The main benefit of ORPO over RLHF with PPO and over DPO is the removal of the additional preference alignment training step. Instead preference alignment is performed together with domain adaptation, reducing both memory requirements and FLOPs per batch.

In addition, the team evaluated the impact on performance of the outcome model itself following ORPO, and showed that ORPO improves the model performance.
The team trained multiple models ranging in size from 125M to 7B, including Mistral-7B and LLama-2-7B models, comparing SFT, PPO, DPO and ORPO. The prompts and preference pairs for answers (chosen, rejected) were taken from the Anthropic HH-RLHF dataset and the Binarized UltraFeedback dataset. Evaluation was done with AlpacaEval (single turn instruction following - to test the domain adaptation) and MT-Bench benchmarks (to test preference optimization), using GPT-4 variants as evaluators (section 5.2).
First, results show that mixing in the preference alignment with the domain adaptation in a single step using ORPO did not adversely impact but actually improved domain adaptation:

Model Name	Size	$\text{AlpacaEval}_{1.0}$	$\text{AlpacaEval}_{2.0}$
Phi-2 + SFT	2.7B	48.37% (1.77)	0.11% (0.06)
Phi-2 + SFT + DPO	2.7B	50.63% (1.77)	0.78% (0.22)
Phi-2 + ORPO *(Ours)*	2.7B	**71.80% (1.59)**	**6.35% (0.74)**
Llama-2 Chat *	7B	71.34% (1.59)	4.96% (0.67)
Llama-2 Chat *	13B	81.09% (1.38)	7.70% (0.83)
Llama-2 + ORPO *(Ours)*	7B	**81.26% (1.37)**	**9.44% (0.85)**
Zephyr (α) *	7B	85.76% (1.23)	8.35% (0.87)
Zephyr (β) *	7B	90.60% (1.03)	10.99% (0.96)
Mistral-ORPO-α *(Ours)*	7B	87.92% (1.14)	11.33% (0.97)
Mistral-ORPO-β *(Ours)*	7B	**91.41% (1.15)**	**12.20% (0.98)**

Table 1: Table of instruction-following abilities of each checkpoint measured through AlpacaEval. While clearly showing the improvements in instruction-following abilities after training with ORPO, it is notable that ORPO models exceed RLHF or DPO models of Llama-2 and Mistral (* indicates the results from the official leaderboard.)

Then, the team assessed the win rate of ORPO over other preference alignment methods (RLHF with PPO, DPO), showing consistent wins for ORPO for larger models:

ORPO vs	SFT	+DPO	+PPO
OPT-125M	73.2 (0.12)	48.8 (0.29)	71.4 (0.28)
OPT-350M	80.5 (0.54)	50.5 (0.17)	85.8 (0.62)
OPT-1.3B	69.4 (0.57)	57.8 (0.73)	65.7 (1.07)

Table 3: Average win rate (%) and its standard deviation of ORPO and standard deviation over other methods on **UltraFeedback** dataset for three rounds. Sampling decoding with a temperature of 1.0 was used.

ORPO vs	SFT	+DPO	+PPO
OPT-125M	84.0 (0.62)	41.7 (0.77)	66.1 (0.26)
OPT-350M	82.7 (0.56)	49.4 (0.54)	79.4 (0.29)
OPT-1.3B	78.0 (0.16)	70.9 (0.52)	65.9 (0.33)

Table 2: Average win rate (%) and its standard deviation of ORPO and standard deviation over other methods on **HH-RLHF** dataset for three rounds. Sampling decoding with a temperature of 1.0 was used on the test set.

NOTE: I did not include mention of an additional result - the reward distribution leads to higher expected reward for ORPO-trained models. (Figure 5 in the article). It is also shown that the likelihood of the rejected answer drops during training while the likelihood of the accepted answer keeps increasing, showing that preference alignment works simultaneously without negatively affecting domain adaptation.

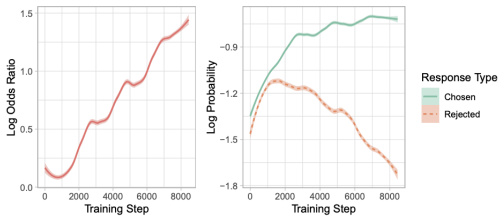

Figure 7: Average log-likelihood for chosen and rejected responses and log odds ratio per batch. The odds consistently increase during training with ORPO.

The team released source code for the training method and it has been since incorporated into the HuggingFace TRL library.