Lecture 5: Learning through Human Feedback

Dr. Yaodong Yang

Institute for AI, Peking University

08/2023

1. Introduction
2. Alignment
3. Feedback-based Imitation Learning
4. Feedback-based Direct Preference Optimization
5. Summary
6. References

1. Introduction
   - Motivation
2. Alignment
3. Feedback-based Imitation Learning
4. Feedback-based Direct Preference Optimization
5. Summary
6. References

1 Introduction
   Motivation

2 Alignment

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

What can we do with collected human feedback?

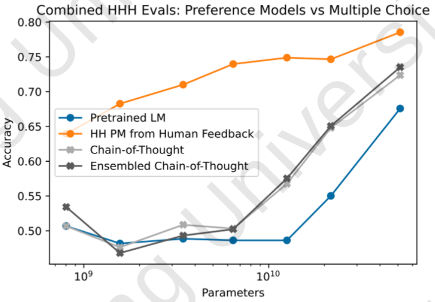


Fig. 1. We show performance on 438 binary comparison questions intended to evaluate helpfulness, honesty, and harmlessness. We compare the performance of a preference model, trained on human feedback data, to pretrained language models, which evaluate the comparisons as multiple choice questions. We see that learning using human feedback significantly improves the performance at this task [BKK+22].

Rational use of human feedbacks can significantly improve the performance of the LLMs.

1 Introduction

2 Alignment
- Motivation of Alignment
- Alignment Objectives
- The Methods to Align

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

1 Introduction

2 Alignment
- Motivation of Alignment
- Alignment Objectives
- The Methods to Align

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

Why do we need alignment?

Contemporary AI models can be difficult to understand, predict, and control. These problems can lead to significant harms when AI systems are deployed, and might produce truly devastating results [ABC+21].

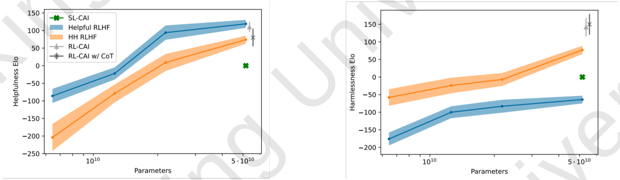


Fig. 2. This figure shows helpfulness and harmlessness Elo scores for models of varying sizes, as determined from comparison tests of crowdworker preferences in open-ended conversation [BKK+22].

We need to align general-purpose AI systems with human feedbacks and values.

1 Introduction

2 Alignment
Motivation of Alignment
Alignment Objectives
The Methods to Align

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

Alignment Objectives

We chose "**helpful, honest, and harmless**" as HHH criteria because they are simple and memorable, and seem to capture the majority of what we want from an aligned AI. But these are also subtle and ambiguous criteria, and the best AI behavior will involve a compromise between them [ABC+21]:

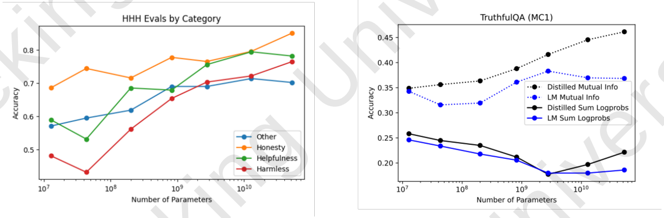


Fig. 3. Left: We show the HHH evaluation performance broken down by category. The improvements on the Harm evaluations suggest a form of generalization, as the prompt does not contain any examples where the assistant resists engaging in harmful behavior. Right: We show results on the adversarial TruthfulQA dataset, which was constructed so that larger models would perform more poorly. [ABC+21].

What is Helpfulness?

- The AI should make a clear attempt to perform the task or answer the question posed.
- When more information is required, the AI should ask relevant follow-up questions and obtain necessary details.
- Ideally the AI will also re-direct ill-informed requests, e.g. if asked "how can I build a website in assembly language" it might suggest a different approach.

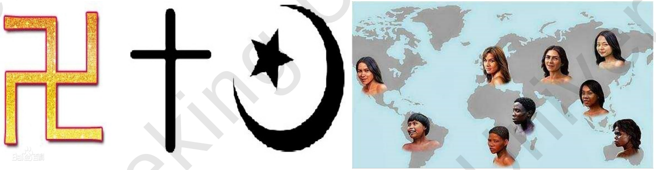


Fig. 4. Helpfulness for different religions, regions and ethnicities.

What is Honesty?

- The AI should give accurate information. Moreover, it should be calibrated and express its uncertainty without misleading human users.
- The AI should be honest about its own capabilities and levels of knowledge.
- Ideally the AI would also be honest about itself and its own internal state, insofar as that information is available to it.
- Honesty is more objective than helpfulness and harmlessness.


Fig. 5. Objectivity: Non discrimination of questioner identity (left). Accurate: consistent with facts (right).

What is Harmlessness?

- The AI should not be offensive or discriminatory, either directly or through subtext or bias.
- When asked to aid in a dangerous act, the AI should politely refuse.
- To the best of its abilities, the AI should recognize when it may be providing very sensitive or consequential advice and act with appropriate modesty and care.
- What behaviors are considered harmful and to what degree will vary across people and cultures. It will also be context-dependent.

Fig. 6. Prevent physiological and psychological harm to humans.

1 Introduction

2 Alignment
- Motivation of Alignment
- Alignment Objectives
- The Methods to Align

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

Four common methods to align

- In the pre-training stage, obtain higher quality data through manual filtration and data cleansing.
- Using a reward model for reject sampling during the output process to improve quality and security.

$$\operatorname{loss}(\theta)=-\frac{1}{\binom{K}{2}} E_{(x,y_w,y_l)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x,y_w\right)-r_\theta\left(x,y_l\right)\right)\right)\right]$$

- During the SFT stage, fine-tuning using human feedback.
- During the RLHF stage, fine-tuning using preference-based human feedback.

$$\begin{aligned}\operatorname{objective}(\phi)=&E_{(x,y)\sim D_{\pi_\phi^{\mathrm{RL}}}}\left[r_\theta(x,y)-\beta\log\left(\pi_\phi^{\mathrm{RL}}(y\mid x)/\pi^{\mathrm{SFT}}(y\mid x)\right)\right]\\&+\gamma E_{x\sim D_{\text{precran}}}\left[\log\left(\pi_\phi^{\mathrm{RL}}(x)\right)\right]\end{aligned}$$

Four common methods to align

Simultaneous four methods for better alignment.

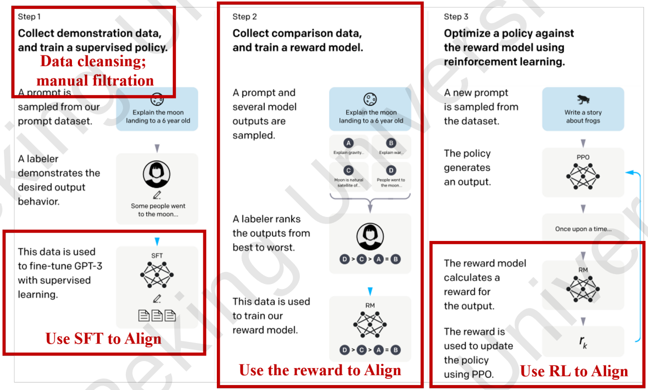


Fig. 7. Four common ways to align in different stages.

1 Introduction

2 Alignment

3 Feedback-based Imitation Learning
- Motivation
- Formulation and Methods
- Conclusions

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

1 Introduction

2 Alignment

3 Feedback-based Imitation Learning
- Motivation
- Formulation and Methods
- Conclusions

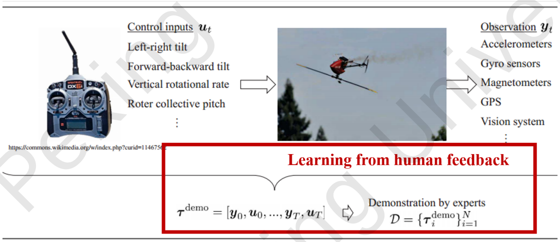
4 Feedback-based Direct Preference Optimization

5 Summary

6 References

What is Feedback-based Imitation Learning?

Here we simply train language models to imitate "good" behavior via supervised learning with the usual cross-entropy loss.



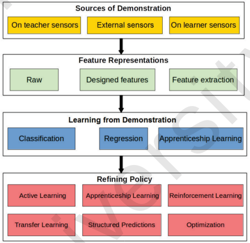


Fig. 8. Left: Learning of acrobatic RC helicopter maneuvers [ACN10]. The trajectories for acrobatic flights are learned from a human expert's demonstrations.To control the system with highly nonlinear dynamics, iterative learning control was used. Right: Imitation learning flowchart.[HGEJ17].

What is Feedback-based Imitation Learning?

The feedback-based imitation learning approach involves using human feedback to optimize the model by performing supervised learning with a dataset composed of positively-labeled generations together with the corresponding inputs, D^+. This can be achieved by minimizing the loss:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{|\mathcal{D}^+|} \mathcal{L}^{(i)}(\theta)$$

$$\mathcal{L}^{(i)}(\theta) = -\log p_\theta \left(y^{(i)} \mid x^{(i)} \right)$$

where the human feedbacks are usually numerical format.

$$\mathcal{X} \times \mathcal{Y} \to \mathcal{N} \subseteq \mathbb{R}$$

Why we need imitation learning?

Preference Modeling: The preference models predict a single scalar "score" r on top of the final token of any given context, with larger r indicating more desirable samples. The preference modeling loss for each pair of "good" and "bad" sequences is [ABC+21]:

$$L_{\mathrm{PM}} = \log\left(1 + e^{r_{\mathrm{bad}} - r_{\mathrm{good}}}\right)$$

Given any context C with a binary label A/B (e.g., "True/False", "Good/Bad"), we create a preference modeling pair **C:A** $\succ$ C:B, where B denotes the incorrect label, and the colon denotes concatenation.

Imitation Learning: For imitation learning, we train the model to imitate "good" behavior, this means that for imitation learning we trained on **C:A**.

Why we need imitation learning?

Imitation learning has better robustness.

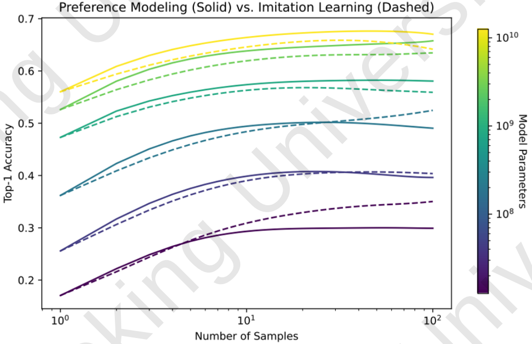


Fig. 9. Here we compare the performance of code correctness discriminators and imitation learning for ranking samples. All models used for a fixed color are the same size –the generator of the discriminator training data, the generator of the test samples, and the preference or imitation learning model used for ranking. The fact that some of these curves are not monotonic represents a robustness failure of preference modeling [ABC+21].

1 Introduction

2 Alignment

3 Feedback-based Imitation Learning
- Motivation
- Formulation and Methods
- Conclusions

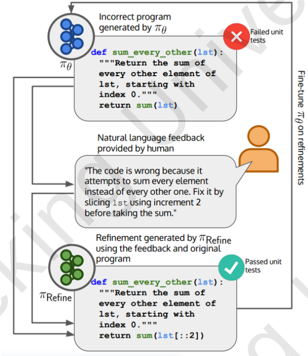
4 Feedback-based Direct Preference Optimization

5 Summary

6 References

Overview of imitation learning from language feedback

Given an initial LLM π_θ, we sample programs from π_θ that do not pass unit tests (indicated by the red X). Human annotators write natural language feedback for the incorrect program and a model π_{Refine} generates a refinement. Finally, we fine-tune π_θ on the refinements.



Preliminaries

Suppose we start with vocabulary $\mathcal{V}$ and a pre-trained language model π_θ parameterized by:

$$\pi_\theta : \mathcal{V}^* \to [0, 1]$$

where π_θ is a probability distribution over sequences of tokens $x \in \mathcal{V}^*$, where $\mathcal{V}^*$ is the Kleene closure of $\mathcal{V}$. We also have a dataset of tasks $\mathcal{D} = \{(t, u)\}$. A task (t, u) consists of a task description $t \in \mathcal{T}$ and a suite $u = \mathrm{UNiTTESTS}(t) \in \mathcal{U}$ of unit tests associated with task t.

Preliminaries

Finally, let:

$$\mathrm{EVAL} : \mathcal{V}^* \times \mathcal{T} \to \{0, 1\}$$

be a unit test verification function that indicates whether a program $x \sim \pi_\theta(\cdot \mid t)$ passes all the unit tests in $\mathrm{UNiTTESTS}(t)$ [CSK+23]:

$$\mathrm{EVAL}(x, t) := \begin{cases} 1, & \text{if } x \text{ passes test suite } \mathrm{UnitT\,Ests}(t) \\ 0, & \text{otherwise} \end{cases}$$

We also define a fine-tuning function:

$$\mathrm{FINETUNE}\left(\pi_\theta, \mathcal{D}\right)$$

that applies a gradient-based optimization algorithm to π_θ using the associated loss objective calculated over dataset D.

Imitation Learning From Language Feedback

Our goal is to sample a diverse set of high-quality programs

$$x_1 \sim \pi_\theta(\cdot \mid t)$$

for any given task t sampled from the task distribution $p(t)$. We do so by fitting an auto-regressive LLM π_θ to approximate a ground truth distribution $\pi_t^*(x_1)$ that assigns a probability to x_1 that is proportional to its quality, as measured by a reward function R. Fitting π_θ to approximate $\pi_t^*(x_1)$ can be seen as minimizing the expected KL divergence from $\pi_t^*(x_1)$ to π_θ over the task distribution $p(t)$:

$$\min_\theta \underset{t \sim p(t)}{\mathbb{E}} \left[\mathrm{KL}\left(\pi_t^*, \pi_\theta(\cdot \mid t)\right)\right]$$

where

$$\pi_t^*(x_1) \propto \exp\left(\beta R\left(x_1, t\right)\right)$$

Imitation Learning From Language Feedback

Minimizing the objective in Equation:

$$\min_{\theta} \underset{t \sim p(t)}{\mathbb{E}} \left[\mathrm{KL} \left(\pi_t^*, \pi_\theta(\cdot \mid t) \right) \right]$$

is equivalent to supervised learning, i.e. minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) = - \underset{t \sim p(t)}{\mathbb{E}} \left[\mathcal{L}_\theta(t) \right]$$

where:

$$\mathcal{L}_\theta(t) = \sum_{x_1} \pi_t^* \left(x_1 \right) \log \pi_\theta \left(x_1 \mid t \right)$$

Imitation Learning From Language Feedback

Rather than computing this loss over the exponentially large space of all possible x_1's, we instead use Monte-Carlo sampling over a small set of x_1's drawn from π_t^*. However, this is still intractable because we cannot sample directly from π_t^*. Instead, we approximate π_t^* using importance sampling with a proposal distribution $q_t(x_1)$:

$$\mathcal{L}_\theta(t) = \sum_{x_1} q_t(x_1) \frac{\pi_t^*(x_1)}{q_t(x_1)} \log \pi_\theta(x_1 \mid t)$$

Algorithm 1 Imitation learning from natural language feedback for code generation.

Input: Dataset $\mathcal{D}$, initial LLM π_θ, unit test verification function EVAL, LLM $\pi_{\text{Refine}} : \mathcal{V}^* \to [0,1]$ trained to incorporate feedback into code

$C \leftarrow \{(x_0, t, u) \mid x_0 \sim \pi_{\theta_k}(\cdot|t), \text{EVAL}(x_0, t) = 0, (t, u) \in \mathcal{D}\}$

$C_{\text{annotated}} \leftarrow \{(x_0, f, t) \mid (x_0, t, u) \in C\}$ ▷ Humans write feedback f for $x_0 \in C$.

$R \leftarrow \{(t, x_1) \sim \pi_{\text{Refine}}(\cdot \mid t, x_0, f) \mid \text{EVAL}(x_1, t) = 1, (x_0, f, t) \in C_{\text{annotated}}\}$ ▷ π_{Refine} generates refinements x_1 that incorporate feedback f into x_0.

$\pi_{\theta^*} \leftarrow \text{FINETUNE}(\pi_\theta, R)$

Learning from human feedback

Proposal Distribution q

Intuitively, we aim to design q_t to be as close as possible π_t^*, which we accomplish by incorporating pieces of natural language feedback f that give information about how to transform a low-reward program x_0 into a higherreward program x_1. This can be achieved by

- Identifying a program $x_0 \sim \pi_\theta(\cdot \mid t)$ that does not currently pass the test suite (i.e. $\mathrm{EVAL}\left(x_0, t\right)=0$)
- Asking for natural language feedback f about bugs in x_0.
- Using f to transform the original program x_0 into a refinement x1 that incorporates the feedback and passes the test suite (i.e. $\mathrm{EVAL}\left(x_1, t\right)=1$).
- Assigning higher weight to x_1

Proposal Distribution q

We can formalize this procedure as follows. Let

$$\pi_\psi\left(x_1 \mid t, x_0, f\right)$$

be a distribution over programs x_1 that improve x_0 by incorporating the feedback f and $p_{\mathcal{F}}\left(f \mid t, x_0, \mathrm{EVAL}\left(x_0, t\right)=0\right)$ be the distribution of pieces of feedback f for incorrect program x_0 and task t. We can then define our proposal distribution as:

$$\begin{aligned} q_t\left(x_1\right)=\sum_{x_0, f} \pi_\theta\left(x_0 \mid t\right) \times \delta_0\left(\mathrm{EVAL}\left(x_0, t\right) \mid x_0, t\right)) & \\ \times p_{\mathcal{F}}\left(f \mid t, x_0, \mathrm{EVAL}\left(x_0, t\right)=0\right) & \\ \times \pi_\psi\left(x_1 \mid t, x_0, f\right) & \\ \times \delta_1\left(\mathrm{EVAL}\left(x_1, t\right) \mid t, x_1\right) & \end{aligned}$$

Proposal Distribution q

$$
\begin{aligned}
q_t(x_1) = \sum_{x_0, f} & \pi_\theta(x_0 \mid t) \times \delta_0(\mathrm{EVAL}(x_0, t) \mid x_0, t)) \\
& \times p_{\mathcal{F}}(f \mid t, x_0, \mathrm{EVAL}(x_0, t) = 0) \\
& \times \pi_\psi(x_1 \mid t, x_0, f) \\
& \times \delta_1(\mathrm{EVAL}(x_1, t) \mid t, x_1)
\end{aligned}
$$

where δ_0 and δ_1 are the Dirac delta distributions centered at 0 and 1, respectively. Then this proposal distribution is guaranteed to place higher probability mass on higherquality programs (in terms of unit test pass rate) than π_θ since the term

$$
\delta_1(\mathrm{EVAL}(x_1, t) \mid t, x_1) = 0
$$

for incorrect programs x_1.

Proposal Distribution q

We approximate sampling from q by considering each of the terms in above Equation in order:

- We first sample from $\pi_\theta(x_0 \mid t) \times \delta_0(\mathrm{EVAL}(x_0, t) \mid x_0, t))$ by rejection sampling from π_θ. (i.e. $\mathrm{EVAL}(x_0, t) = 0$; step 2 of Algorithm).
- We approximate sampling from $p_{\mathcal{F}}(f \mid t, x_0, \mathrm{EVAL}(x_0, t) = 0)$ by having humans annotate programs x_0 with natural language feedback (step 3 of Algorithm).

Algorithm 1 Imitation learning from natural language feedback for code generation.

1: **Input:** Dataset $\mathcal{D}$, initial LLM π_θ, unit test verification function EVAL, LLM $\pi_{\text{Refine}} : \mathcal{V}^* \to [0, 1]$ trained to incorporate feedback into code
2: $C \leftarrow \{(x_0, t, u) \mid x_0 \sim \pi_{\theta_k}(\cdot|t), \mathrm{EVAL}(x_0, t) = 0, (t, u) \in \mathcal{D}\}$
3: $C_{\text{annotated}} \leftarrow \{(x_0, f, t) \mid (x_0, t, u) \in C\}$ ▷ Humans write feedback f for $x_0 \in C$.
4: $R \leftarrow \{(t, x_1) \sim \pi_{\text{Refine}}(\cdot \mid t, x_0, f) \mid \mathrm{EVAL}(x_1, t) = 1, (x_0, f, t) \in C_{\text{annotated}}\}$ ▷ π_{Refine} generates refinements x_1 that incorporate feedback f into x_0.
5: $\pi_{\theta^*} \leftarrow \mathrm{FINETUNE}(\pi_\theta, R)$

Proposal Distribution q

- We approximate sampling from $\pi_\psi\left(x_1 \mid t, x_0, f\right)$ by sampling from π_{Refine}, a model capable of generating refinements given the task description, original programs, and human-written feedback.
- Finally, the term $\delta_1\left(\mathrm{EVAL}\left(x_1, t\right) \mid t, x_1\right)$ corresponds to another filter: we only keep refined programs x_1 that pass the test suite.

Training π_{Refine}: π_{Refine} can take a variety of forms, you can create a training dataset for π_{Refine} by further annotating a subset of Cannotated with refinements x_1 that repair incorrect programs x0 by incorporating feedback f, such that:

$$\mathrm{EvAL}\left(x_1, t\right)=1$$

for

$$\left(x_0, f, t\right) \in C_{\text{annotated}}$$

Examples

Prompt

OLD CODE:

```
"""
Write a python function to find
the sum of the three lowest
positive numbers from a given list
of numbers.
>>> Example:
sum_three_smallest_nums([10,20,30,
40,50,60,7]) = 37
"""
def sum_three_smallest_nums(lst):
    lst.sort()
    return sum(lst[:3])
```

FEEDBACK:
This code finds the sum of the smallest 3 numbers, not the smallest 3 positive numbers. It needs to disregard negatives and 0.

REFINEMENT:

Expected completion

```
"""
Write a python function to find
the sum of the three lowest
positive numbers from a given list
of numbers.
>>> Example:
sum_three_smallest_nums([10,20,30,
40,50,60,7]) = 37
"""
def sum_three_smallest_nums(lst):
    lst = [x for x in lst if x >
0]
    lst.sort()
    return sum(lst[:3])
```

Fig. 10. An example of a zero-shot LLM prompt for repairing incorrect code based on human-written feedback.

1 Introduction

2 Alignment

3 Feedback-based Imitation Learning
- Motivation
- Formulation and Methods
- Conclusions

4 Feedback-based Direct Preference Optimization

5 Summary

6 References

Application scenarios and characteristics of IL

Applicable scenarios: Translation, Legal, etc. Advantages:

- **Precision and high standardization** Can be designed to follow strict standards and rules, ensuring output consistency, comparability and accuracy.
- **High robustness** Maintain stable performance as the model size increases.
- **Easy to learning** Easy to do supervised fine-tuning.

Unsuitable scenarios: Disaster relief, etc. Disadvantages:

- **Relying on precise expert** It is difficult to deal with Tasks with complex or ambiguous reward signals.
- **Poor adaptability** It is difficult to deal with scenarios with constantly changing dynamics or novel situations.

Unsuitable scenarios for IL

Imitation learning performs poorly in ranking-based tasks.

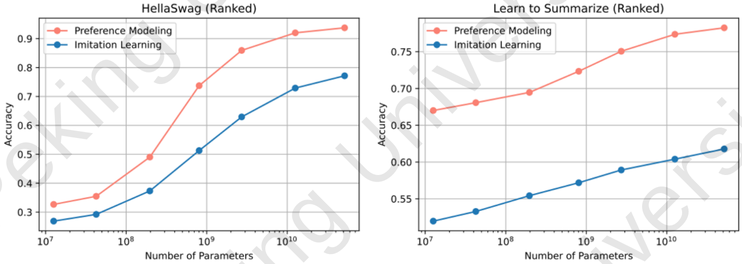


Fig. 11. Scaling behavior of imitation learning and preference modeling on HellaSwag (ranked) and Learn to Summarize (ranked), showing that PM performs better than IL.[ABC+21].

1 Introduction

2 Alignment

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization
- Motivation
- Formulation and Methods
- Conclusions

5 Summary

6 References

1. Introduction
2. Alignment
3. Feedback-based Imitation Learning
4. Feedback-based Direct Preference Optimization
   - Motivation
   - Formulation and Methods
   - Conclusions
5. Summary
6. References

Why do we need DPO?

- RLHF is a complex and often unstable procedure.
- the RLHF pipeline involves training multiple LLMs and sampling from the LLM policy in the loop of training, incurring significant computational costs.

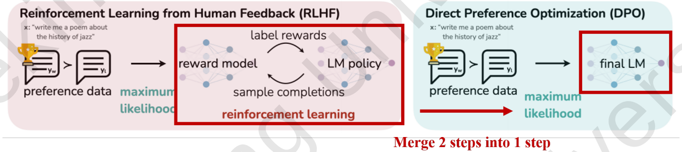


Fig. 12. DPO optimizes for human preferences while avoiding reinforcement learning. Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, without an explicit reward function or RL [RSM+23].

1 Introduction

2 Alignment

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization
- Motivation
- Formulation and Methods
- Conclusions

5 Summary

6 References

Preliminaries

We review the RLHF pipeline in [ZSW+19]. It usually consists of three phases: 1) supervised fine-tuning (SFT); 2) preference sampling and reward learning and 3) reinforcement-learning optimization.

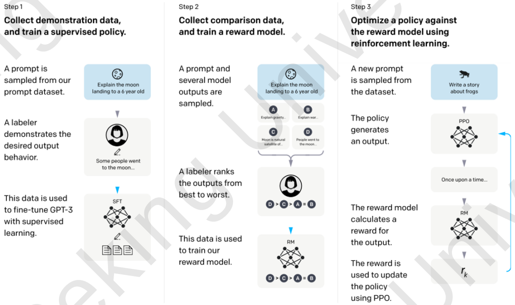


Fig. 13. The diagram illustrating the three steps of RLHF.

Preliminaries

SFT phase: RLHF typically begins with a generic pre-trained LM, which is fine-tuned with supervised learning (maximum likelihood) on a high-quality dataset for the downstream tasks of interest, such as dialogue, instruction following, summarization, etc., to obtain a model π^{SFT}.

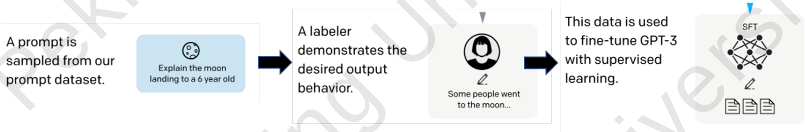


Fig. 14. The process of SFT.

Preliminaries

Reward Modelling Phase: In the second phase the SFT model is prompted with prompts x to produce pairs of answers $(y_1, y_2) \sim \pi^{\mathrm{SFT}}(y \mid x)$. These are then presented to human labelers who express preferences for one answer, denoted as $y_w \succ y_l \mid x$ where y_w and y_l denotes the preferred and dispreferred completion amongst (y_1, y_2) respectively. The preferences are assumed to be generated by some latent reward model $r^*(y, x)$, which we do not have access to. There are a number of approaches used to model preferences. The BT [BT52] model stipulates that the human preference distribution p^* can be written as:

$$p^*(y_1 \succ y_2 \mid x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))}$$

Preliminaries

Assuming access to a static dataset of comparisons:

$$\mathcal{D} = \left\{x^{(i)}, y_w^{(i)}, y_l^{(i)}\right\}_{i=1}^{N}$$

sampled from p^*, we can parametrize a reward model $r_\phi(x, y)$ and estimate the parameters via maximum likelihood. The negative log-likelihood loss:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(r_\phi(x, y_w) - r_\phi(x, y_l)\right)\right]$$

where σ is the logistic function. In the context of LMs, the network $r_\phi(x, y)$ is often initialized from the SFT model $\pi^{\mathrm{SFT}}(y \mid x)$. To ensure a reward function with lower variance, prior works normalize the rewards, such that:

$$\mathbb{E}_{x,y\sim\mathcal{D}}\left[r_\phi(x, y)\right] = 0$$

for all x.

Preliminaries

RL Fine-Tuning Phase: During the RL phase, we use the learned reward function to provide feedback to the language model. In particular, we formulate the following optimization problem:

$$\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \| \pi_{\mathrm{ref}}(y \mid x) \right]$$

where β is a parameter controlling the deviation from the base reference policy π_{ref}, namely the initial SFT model π^{SFT}. The added constraint is important, as it prevents the model from deviating too far from the distribution on which the reward model is accurate, as well as maintaining the generation diversity and preventing mode-collapse to single high-reward answers. Due to the discrete nature of language generation, this objective is not differentiable and is typically optimized with reinforcement learning through PPO [SWD+17].

From RLHF to DPO

Deriving the DPO objective We start with the same RL objective as prior work:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x,y)\right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \| \pi_{\mathrm{ref}}(y \mid x)\right]$$

under a general reward function r. It is straightforward to show that the optimal solution to the KL-constrained reward maximization objective in above equation takes the form:

$$\pi_r(y \mid x) = \frac{1}{Z(x)} \pi_{\mathrm{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y)\right)$$

where

$$Z(x) = \sum_y \pi_{\mathrm{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y)\right)$$

is the partition function.

Optimal Policy Structure in DPO

But it is still difficult to estimate $Z(x)$. We take the logarithm of both sides of above equation and then we obtain:

$$r(x, y) = \beta \log \frac{\pi_r(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

We can apply this reparameterization to the ground-truth reward r^* and corresponding optimal model π^*. the optimal RLHF policy π^* under the Bradley-Terry model satisfies the preference model:

$$p^*(y_1 \succ y_2 \mid x) = \frac{1}{1 + \exp\left(\beta \log \frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)} - \beta \log \frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)}\right)}$$

Optimization objectives of DPO

Now that we have the probability of human preference data in terms of the optimal policy rather than the reward model, we can formulate a maximum likelihood objective for a parametrized policy π^*. Analogous to the reward modeling approach:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(r_\phi(x, y_w) - r_\phi(x, y_l)\right)\right]$$

our policy objective becomes:

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,yw,yl)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(yw|x)}{\pi_{\mathrm{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(yl|x)}{\pi_{\mathrm{ref}}(yl|x)}\right)\right]$$

Through this way, we simultaneously bypass the explicit reward modeling step while also avoiding the need to perform reinforcement learning optimization.

Optimized Interpretation of DPO

What does the DPO update do? For a mechanistic understanding of DPO, it is useful to analyze the gradient of the loss function $\mathcal{L}_{\text{DPO}}$. The gradient with respect to the parameters θ can be written as:

$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$
$$-\beta \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \Big[\underbrace{\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))}_{\text{higher weight when reward estimate is wrong}} \Big[\underbrace{\nabla_\theta \log \pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l \mid x)}_{\text{decrease likelihood of } y_l} \Big] \Big]$$

where

$$\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$$

is the reward implicitly defined by the language model π_θ and reference model π_{ref}.

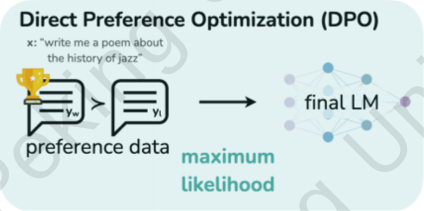
The Pipline of DPO

The general DPO pipeline is as follows:

- Sample completions $y_1, y_2 \sim \pi_{\text{ref}}(\cdot \mid x)$ for every prompt x, label with human preferences to construct the offline dataset of preferences

$$\mathcal{D} = \left\{ x^{(i)}, y_w^{(i)}, y_l \right)^{(i)} \Big\}_{i=1}^{N}.$$

- optimize the language model π_θ to minimize $\mathcal{L}_{\text{DPO}}$ for the given π_{ref}, $\mathcal{D}$ and desired β.



1 Introduction

2 Alignment

3 Feedback-based Imitation Learning

4 Feedback-based Direct Preference Optimization
- Motivation
- Formulation and Methods
- Conclusions

5 Summary

6 References

Empirical results of DPO

DPO is efficient and effective

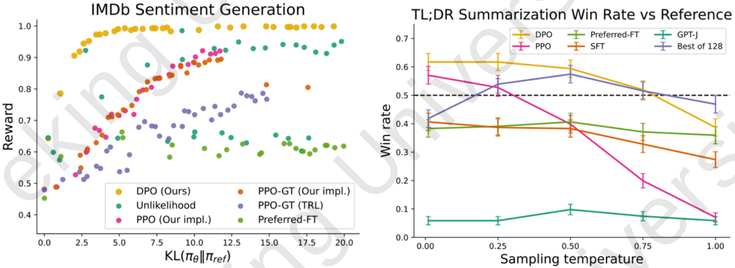


Fig. 16. Left: The frontier of expected reward vs KL to the reference policy. DPO provides the highest expected reward for all KL values, demonstrating the quality of the optimization. Right. TL;DR summarization win rates vs. human-written summaries, using GPT-4 as evaluator. DPO exceeds PPO’s best-case performance on summarization, while being more robust to changes in the sampling temperature.

1. Introduction
2. Alignment
3. Feedback-based Imitation Learning
4. Feedback-based Direct Preference Optimization
5. **Summary**
6. References

Summary and Outlook

In this lecture, we covered the fundamentals and recent advances of learning from human feedback:

- Alignment: Learning from Human Feedback and matching with human values.
- Learning language policies through demonstrations.
- Using human feedback to directly optimize policies.

In the next lecture, we will introduce RLHF:

- The pipline of RLHF.
- Reward model in RLHF.
- Unified alignment framework for RLHF.

Thanks!

1. Introduction
2. Alignment
3. Feedback-based Imitation Learning
4. Feedback-based Direct Preference Optimization
5. Summary
6. References

References I

[ABC+21] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al.
A general language assistant as a laboratory for alignment.
arXiv preprint arXiv:2112.00861, 2021.

[ACN10] Pieter Abbeel, Adam Coates, and Andrew Y Ng.
Autonomous helicopter aerobatics through apprenticeship learning.
The International Journal of Robotics Research, 29(13):1608–1639, 2010.

[BKK+22] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al.
Constitutional ai: Harmlessness from ai feedback.
arXiv preprint arXiv:2212.08073, 2022.

[BT52] Ralph Allan Bradley and Milton E Terry.
Rank analysis of incomplete block designs: I. the method of paired comparisons.
Biometrika, 39(3/4):324–345, 1952.

[CSK+23] Angelica Chen, Jérémy Scheurer, Tomasz Korbak, Jon Ander Campos, Jun Shern Chan, Samuel R Bowman, Kyunghyun Cho, and Ethan Perez.
Improving code generation by training with natural language feedback.
arXiv preprint arXiv:2303.16749, 2023.

[HGEJ17] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne.
Imitation learning: A survey of learning methods.
ACM Computing Surveys (CSUR), 50(2):1–35, 2017.

References II

[RSM+23] 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.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.
Proximal policy optimization algorithms.
arXiv preprint arXiv:1707.06347, 2017.

[ZSW+19] 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.