

Reinforcement learning from human feedback

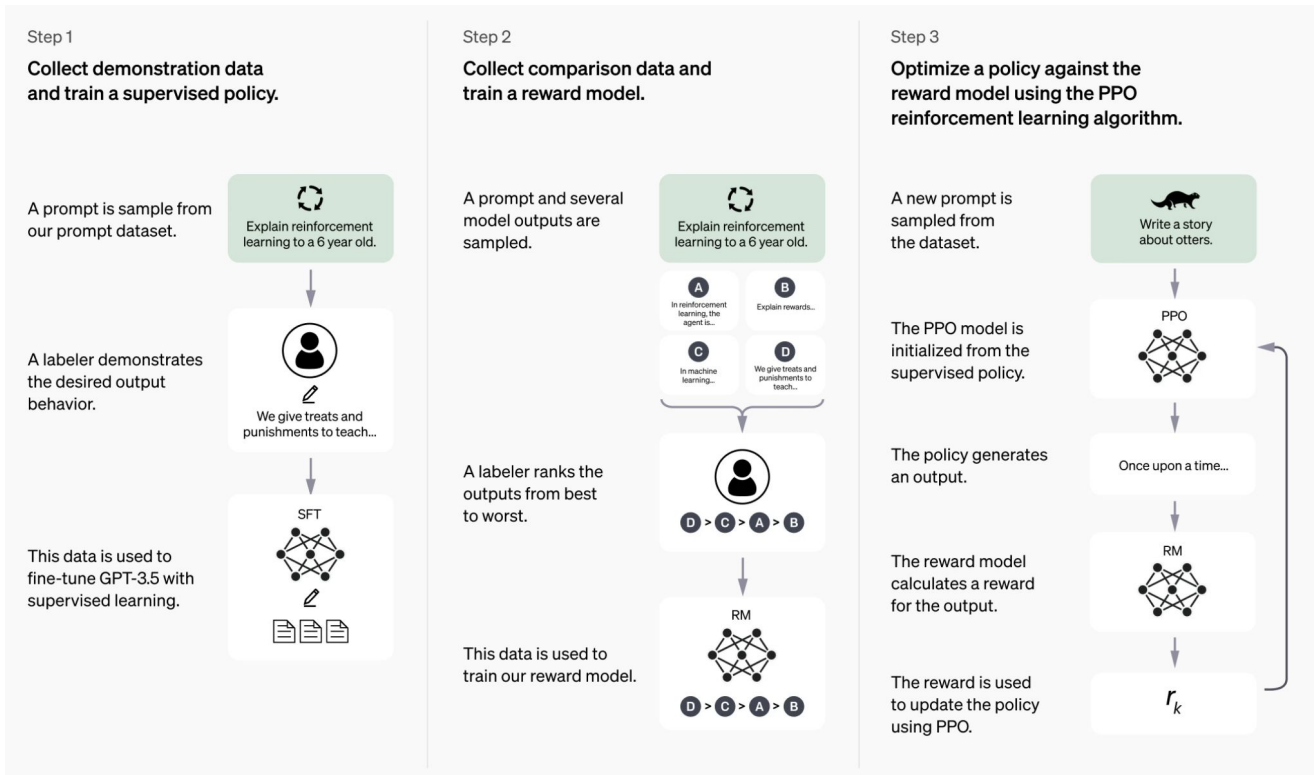
Yunhao

Motivation for RLHF

RLHF is at the forefront of language model training

Recipes for ChatGPT

- Pretraining
- Supervised Fine-tuning
- RLHF



RLHF is at the forefront of language model training

Recipes for ChatGPT

- Pretraining
- Supervised Fine-tuning
- RLHF

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sample from our prompt dataset.



We give treats and punishments to teach...



This data is used to fine-tune GPT-3.5 with supervised learning.



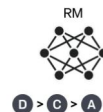
Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.



The PPO model is initialized from the supervised policy.

The policy generates an output.

Once upon a time...



The reward model calculates a reward for the output.

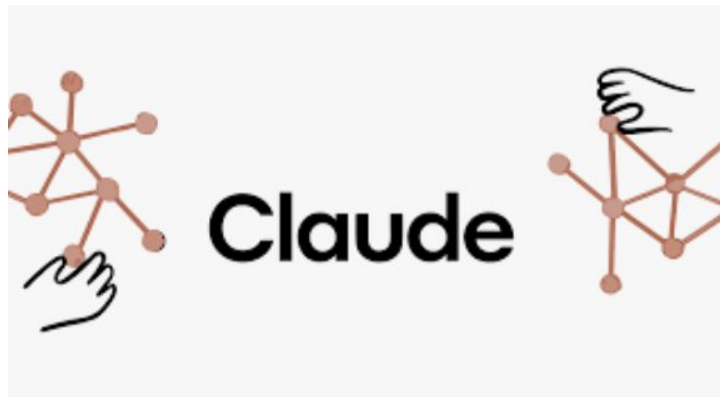
The reward is used to update the policy using PPO.

r_k

RLHF is at the forefront of language model training

RLHF is part of closed source and open source SOTA models

- ChatGPT, Gemini, Claude → RLHF
- Llama → Iterative DPO



RLHF is a fine-tuning recipe for general generative models

The application of RLHF is not limited to language models

- Large language models
- Image, video generation models

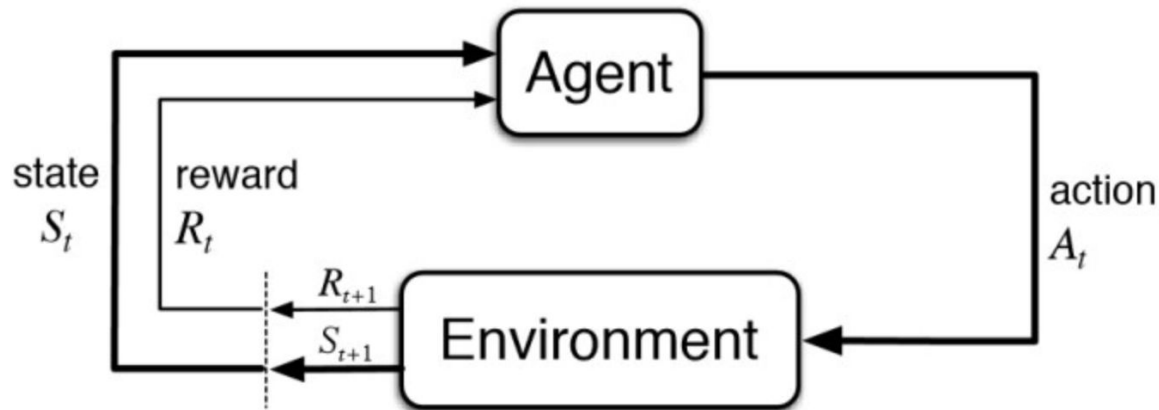
Wherever human can provide feedback, RLHF might be of help...

.

Background on RL

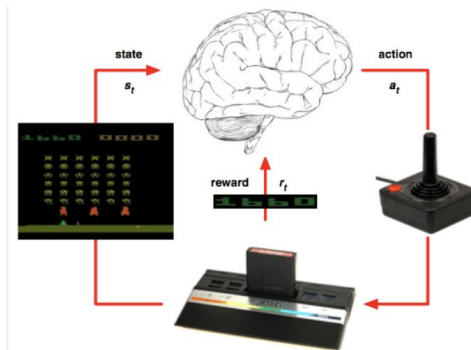
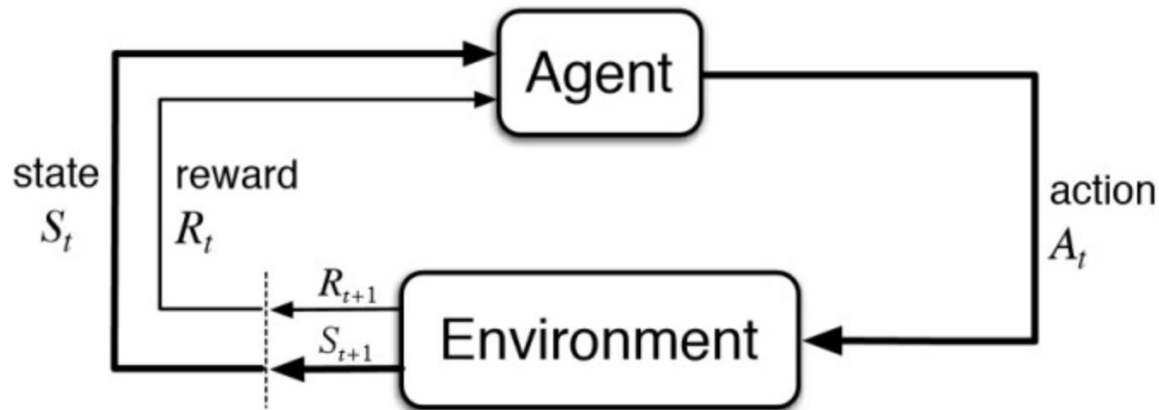
RL: A general framework for sequential decision making

- Agent takes actions $A(t)$ in an environment
- The environment transitions to state $S(t) \rightarrow S(t+1)$
- The environment also produces a reward $R(t)$



RL: A general framework for sequential decision making

- Agent takes actions $A(t)$ in an environment
- The environment transitions to state $S(t) \rightarrow S(t+1)$
- The environment also produces a reward $R(t)$



RL: A general framework for sequential decision making

- The agent's policy defines a mapping from state to distribution over actions
 - "Which button to press when seeing the monster"

$$\pi(A_t | S_t)$$

- Reward function $R_t = R(S_t, A_t)$

RL: A general framework for sequential decision making

- The agent's policy defines a mapping from state to distribution over actions
 - “Which button to press when seeing the monster”

$$\pi(A_t | S_t)$$

- Reward function $R_t = R(S_t, A_t)$

- Objective is to maximize cumulative sum of rewards through policy

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right]$$

The “simplest” algorithm: gradient descent

- Instead of looking for the global optimizer, we do local improvements on the objective via gradient descent

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right] = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [R(\tau)]$$

The “simplest” algorithm: gradient descent

- Instead of looking for the global optimizer, we do local improvements on the objective via gradient descent

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right] = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [R(\tau)]$$

- Policy gradient: derivative of the objective wrt policy parameter

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)]$$

Understanding policy gradient

- Intuition: sample from the current policy
 - Increase probabilities of trajectories proportional to the obtained reward
 - If a reward is higher, the probability of that trajectory also increases more

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)]$$

Understanding policy gradient

- Intuition: sample from the current policy
 - Increase probabilities of trajectories proportional to the obtained reward
 - If a reward is higher, the probability of that trajectory also increases more

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}(\tau)} [R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)]$$

- Policy gradient can be estimated in an unbiased way using samples

$$R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)$$

Stochastic gradient optimization

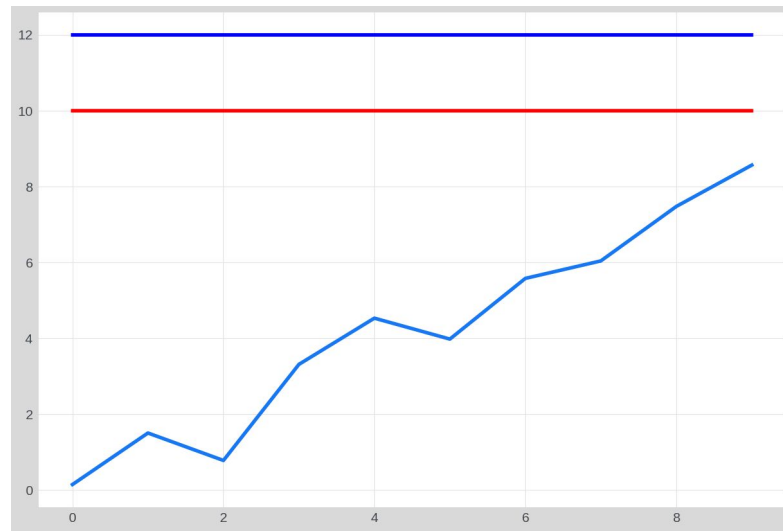
Update parameters using the randomly estimated gradient

$$\theta \leftarrow \theta + R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)$$

Stochastic gradient optimization

Update parameters using the randomly estimated gradient

$$\theta \leftarrow \theta + R(\tau) \nabla_{\theta} \log p_{\theta}(\tau)$$



Improving policy gradient with a baseline

- In practice, policy gradient is computed with sample estimates
 - Unbiased but variance can be high
- Introduce a baseline (an “average”) that measures the relative strength of a sampled reward and trajectory

$$(R(\tau) - V) \nabla_{\theta} \log p_{\theta}(\tau)$$

Improving policy gradient with a baseline

- In practice, policy gradient is computed with sample estimates
 - Unbiased but variance can be high
- Introduce a baseline (an “average”) that measures the relative strength of a sampled reward and trajectory

$$(R(\tau) - V) \nabla_{\theta} \log p_{\theta}(\tau)$$

Think two trajectories provide +10 and +6

- If $V = +8$ we can compute the “update magnitude” as +2 vs. -2

Trajectory gradient -> Step level policy gradient

- So far we have considered a full trajectory as a single entity
 - Not making use of the fact that rewards are accumulated over time, and that actions in the future do not impact rewards in the past
 - Effectively, the problem is an one-step optimization problem (or bandit)

Trajectory gradient -> Step level policy gradient

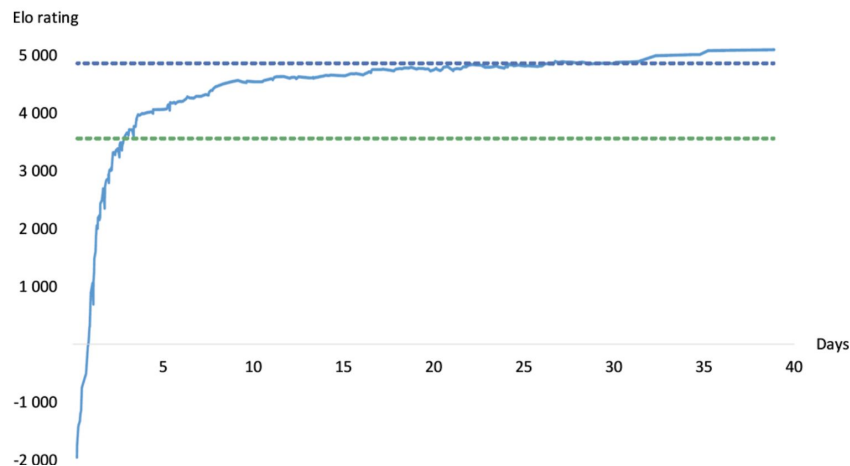
- So far we have considered a full trajectory as a single entity
 - Not making use of the fact that rewards are accumulated over time, and that actions in the future do not impact rewards in the past
 - Effectively, the problem is an one-step optimization problem (or bandit)
- Interestingly, this formulation suffices for RLHF discussion
 - Let's loop back on this point later if we have time

Success stories of RL

In case we have “perfect” rewards, RL systems have achieved super-human performance.

- AlphaGo, AlphaStar, Atari...

This differs significantly from RLHF where perfect rewards are not available.



Background on training language models

Pretraining

- Pretraining differs from SFT but conceptually quite similar
 - Pretraining makes use of data with bigger orders of magnitude (“internet-scale” data)

Pretraining

- Pretraining differs from SFT but conceptually quite similar
 - Pretraining makes use of data with bigger orders of magnitude (“internet-scale” data)
- After pre-training the model knows “everything” but does not know how to pass on such information to humans. When prompting with
 - “Where is the capital of France?”
 - “Where is the capital of UK? Where is the capital of Germany...”

Supervised fine-tuning

- SFT teaches the model to respond in a more human-preferred way
 - Depending on the application: chatbot, companion, tutor, programming assistant...

Supervised fine-tuning

- SFT teaches the model to respond in a more human-preferred way
 - Depending on the application: chatbot, companion, tutor, programming assistant...
- In general, we have some strings we would like to imitate
 - “Where is the capital of France?” -> “Paris.” / “Hi how are you! I think it is Paris.”
 - “What is the weather in London today?” -> “I don’t have real-time information.” / “Let me look for that It is 25 degree celsius today!”
-

Supervised fine-tuning

- Henceforth we will denote $x = \text{prompt}, y = \text{response}$
- SFT aims to maximize

$$\mathbb{E}_{x,y \sim D} [\log \pi_{\theta}(y|x)]$$

Supervised fine-tuning

- Henceforth we will denote $x = \text{prompt}, y = \text{response}$
- SFT aims to maximize

$$\mathbb{E}_{x,y \sim D} [\log \pi_{\theta}(y|x)]$$

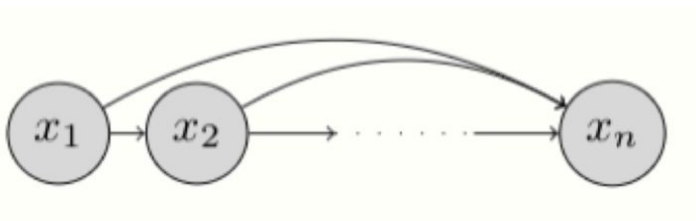
- Where does the data come from?
 - Extract and filter from the internet...
 - Collect from human writers...
 - Collect from powerful models...

Language models consume tokens not strings

- Tokenization: strings are processed in a way that can be consumed by language models
 - “Where is the capital of France?” -> “Hi how are you! I think it is Paris.”
 - [102, 55134, 1,] -> [10, 805, 11133 ...]

Language models consume tokens not strings

- Tokenization: strings are processed in a way that can be consumed by language models
 - “Where is the capital of France?” -> “Hi how are you! I think it is Paris.”
 - [102, 55134, 1,] -> [10, 805, 11133 ...]
- Language model define categorical distributions over such integers \mathbf{x}_i
 - Such a distribution is defined auto-regressively



$$p_{\theta}(\tau) = \prod_{i=1}^n \pi(x_i | x_1 \dots x_{i-1})$$

Reinforcement learning from human feedback

Where does reward come from

Reinforcement learning has always assumed a given reward

- For RLHF, the reward comes from modeling human preference
 - “Write me a poem about history of jazz”
 - “Here is a poem ... (history of jazz)”
 - “Here is a poem ... (history of other stuff)”

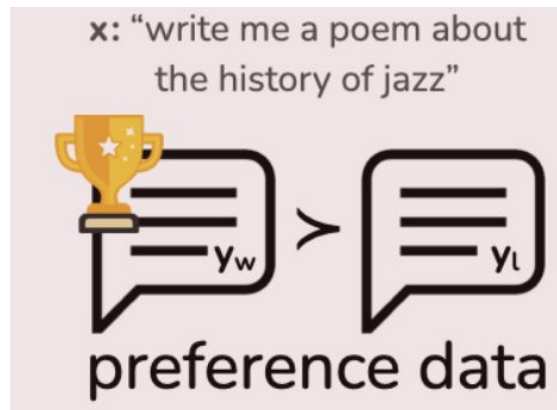


Where does reward come from

Reinforcement learning has always assumed a given reward

- For RLHF, the reward comes from modeling human preference
 - “Write me a poem about history of jazz”
 - “Here is a poem ... (history of jazz)”
 - “Here is a poem ... (history of other stuff)”
- We seek a reward function that captures the preference quality of response y given x

$$r_{\phi}(x, y)$$



Training a reward model from human feedback

A popular assumption is Bradley-Terry model

- Assuming an unknown ground truth reward $r^*(x,y)$
- It defines the probability that one generation is preferred using a logit model

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

Training a reward model from human feedback

A popular assumption is Bradley-Terry model

- Assuming an unknown ground truth reward $r^*(x,y)$
- It defines the probability that one generation is preferred using a logit model

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

- We can estimate the LHS using data from human annotators

Training a reward model from human feedback

Defining a loss that recovers the ground truth unknown reward

- Assuming the BT model is true and there does exist such an unknown reward...
- We can define a maximum likelihood loss below

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

Training a reward model from human feedback

Defining a loss that recovers the ground truth unknown reward

- Assuming the BT model is true and there does exist such an unknown reward...
- We can define a maximum likelihood loss below

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

- In the limit of infinite data and perfect optimization, we can recover \mathbf{r}^*

After the reward is trained... train policy against reward

Reexamine the RL problem now that reward is given

- Regularized policy optimization

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

After the reward is trained... train policy against reward

Reexamine the RL problem now that reward is given

- Regularized policy optimization

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

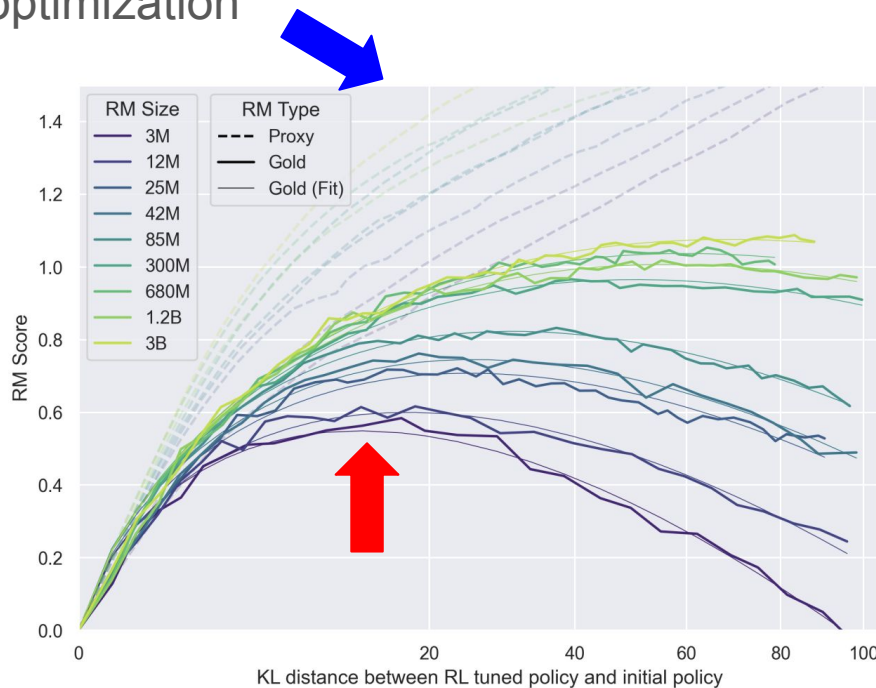


- Objective 1: reward maximization (maximize human preference)
- Objective 2: minimize KL against reference (SFT policy)
 - Reward is always imperfect that we need regularization to prevent model from over-optimization

Goodhart's law: reward model over-optimization

Reward model is always imperfect and we don't want to over-optimize!

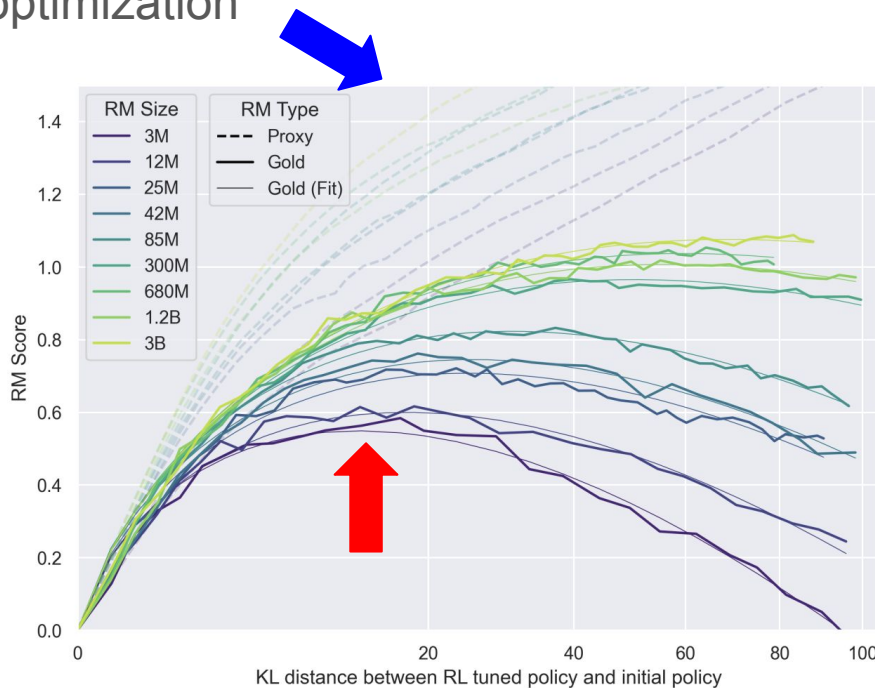
- KL divergence is a way to mitigate over-optimization
- In “classic RL” where reward is perfect
 - We optimize all the way



Goodhart's law: reward model over-optimization

Reward model is always imperfect and we don't want to over-optimize!

- KL divergence is a way to mitigate over-optimization
- In “classic RL” where reward is perfect
 - We optimize all the way
- In a synthetic setup
 - Learned reward always goes up
 - “Golden” reward goes up and down



Can we avoid over-optimization?

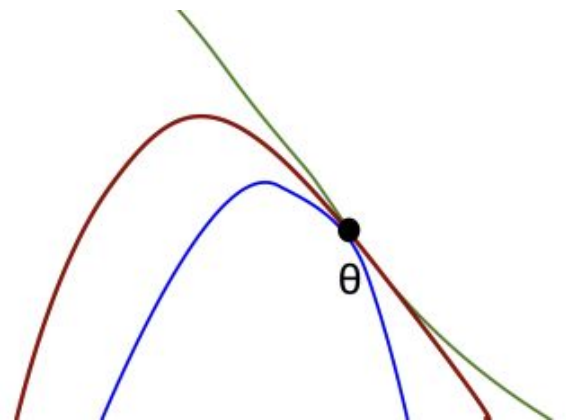
What if we train against human?

- No more over-optimization since we optimize against “golden” reward
- Not practical since human interaction is expensive
 - Policy gradient is very sample inefficient in practice
- A model based approach might incorporate other information
 - Prior knowledge about the problem

Can we avoid over-optimization?

What if we train against human?

- No more over-optimization since we optimize against “golden” reward
- Not practical since human interaction is expensive
 - Policy gradient is very sample inefficient in practice
- A model based approach might incorporate other information
 - Prior knowledge about the problem
- Even human data is not “golden”
 - Green - true user preference (real traffic)
 - Blue - human writer preference (paid writers)
 - Blue - reward model



Summary

- RLHF aims to optimize against human preference
 - Unlike classic RL, RLHF does not have perfect rewards
- RLHF requires learning a reward from human feedback first, before optimizing the policy
 - Due to imperfect rewards, over-optimization is not avoidable
 - Mitigation: training bigger reward models, KL regularization

RLHF without reward
models

Motivation: RLHF is so much more complicated than SFT

- Learning a reward model
 - Try not to overfit a dataset
 - Hard to assess whether a reward model is good because it is an indirect objective
- Policy optimization
 - RL itself is not easy to implement correctly, many things can go wrong
 - RL requires on-policy sampling, which is expensive especially for transformers

Motivation: RLHF is so much more complicated than SFT

- Learning a reward model
 - Try not to overfit a dataset
 - Hard to assess whether a reward model is good because it is an indirect objective
- Policy optimization
 - RL itself is not easy to implement correctly, many things can go wrong
 - RL requires on-policy sampling, which is expensive especially for transformers
- Given a preference dataset, can we just bypass reward modeling?

Connecting optimal policy and reward

The insight lies in the connection between policy and reward

- Regularized policy optimization

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

Connecting optimal policy and reward

The insight lies in the connection between policy and reward

- Regularized policy optimization

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

- Optimal policy takes a specific form

$$\pi_{\theta}^*(y|x) \propto \pi_{\text{ref}}(y|x) \exp(\beta^{-1} r_{\phi}(x, y))$$

A reward learning loss is a policy loss

Let's convert a policy into a reward


- We can convert a policy into a reward

$$\beta \log \pi_{\theta}^*(y|x) / \pi_{\text{ref}}(y|x) - z(x) = r_{\phi}(x, y)$$

A reward learning loss is a policy loss

Let's convert a policy into a reward


- We can convert a policy into a reward

$$\beta \log \pi_{\theta}^*(y|x) / \pi_{\text{ref}}(y|x) - z(x) = r_{\phi}(x, y)$$


A reward learning loss is a policy loss

Let's convert a policy into a reward

- We can convert a policy into a reward

$$\beta \log \pi_{\theta}^*(y|x) / \pi_{\text{ref}}(y|x) - z(x) = r_{\phi}(x, y)$$


- Plugging this into Bradley-Terry loss for reward, yields a loss for policy

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

Direct preference optimization

DPO is defined through the above loss

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

Direct preference optimization

DPO is defined through the above loss

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

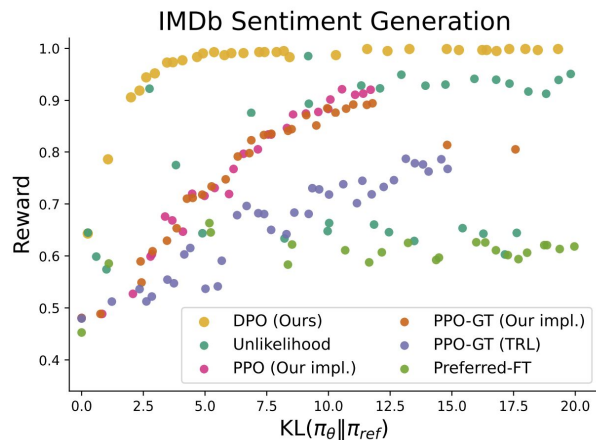
- Increase likelihood of the winning response relative to losing response
- With infinite data and assuming BT model is correct, we will converge to the optimal regularized policy.

Direct preference optimization

DPO is defined through the above loss

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

- Increase likelihood of the winning response relative to losing response
- With infinite data and assuming BT model is correct, we will converge to the optimal regularized policy.



Generalizing the DPO formulation

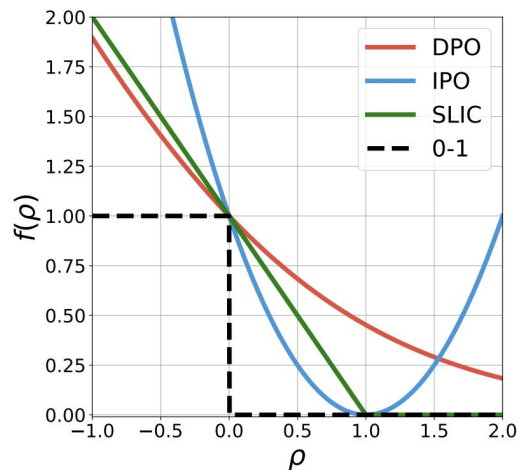
Generalizing the idea of using reward loss as policy loss

- Any reward learning loss \rightarrow policy loss
- Reward model learning is akin to binary classification
 - $r(x, y_1) - r(x, y_2)$ is a measure of relative strength between y_1 and y_2
 - If $r(x, y_1)$ is preferred \rightarrow class 0, otherwise class 1

Generalizing the DPO formulation

Generalizing the idea of using reward loss as policy loss

- Any reward learning loss \rightarrow policy loss
- Reward model learning is akin to binary classification
 - $r(x, y_1) - r(x, y_2)$ is a measure of relative strength between y_1 and y_2
 - If $r(x, y_1)$ is preferred \rightarrow class 0, otherwise class 1
- Convex function class for binary classification
 - Logistic loss \rightarrow DPO
 - Squared loss \rightarrow IPO
 - Hinge loss \rightarrow SLiC



Generalizing the DPO formulation

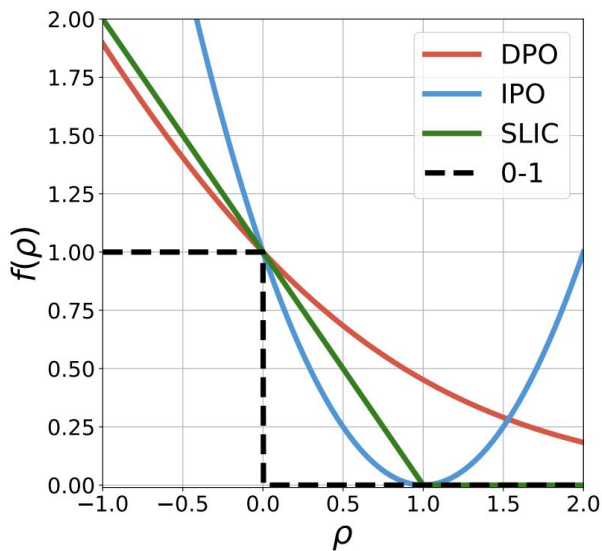
Each convex function for classification \rightarrow reward learning loss \rightarrow policy loss

Generalizing the DPO formulation

Each convex function for classification \rightarrow reward learning loss \rightarrow policy loss

- Convex function class for binary classification
 - Logistic loss \rightarrow DPO
 - Squared loss \rightarrow IPO
 - Hinge loss \rightarrow SLiC

$$\mathbb{E}_{(y_w, y_l) \sim \mu} \left[f \left(\beta \cdot \left(\log \frac{\pi_{\theta}(y_w)}{\pi_{\text{ref}}(y_w)} - \log \frac{\pi_{\theta}(y_l)}{\pi_{\text{ref}}(y_l)} \right) \right) \right]$$



Understanding the trade-offs

In RLHF, we have seen a trade-off between reward maximization and KL minimization, where is such a trade-off now?

- Taylor expanding the convex function

Understanding the trade-offs

In RLHF, we have seen a trade-off between reward maximization and KL minimization, where is such a trade-off now?

- Taylor expanding the convex function $\rho_\theta = \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$

$$\underbrace{\mathbb{E}_{(y_w, y_l) \sim \mu} [f(\beta \rho_\theta)]}_{\text{offline loss}} \approx f(0) + \underbrace{f'(0)\beta \cdot \mathbb{E}_{(y_w, y_l) \sim \mu} [\rho_\theta]}_{\text{preference optimization}} + \underbrace{\frac{f''(0)\beta^2}{2} \cdot \mathbb{E}_{(y_w, y_l) \sim \mu} [\rho_\theta^2]}_{\text{offline regularization}},$$

Understanding the trade-offs

In RLHF, we have seen a trade-off between reward maximization and KL minimization, where is such a trade-off now?

- Taylor expanding the convex function $\rho_\theta = \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$

$$\underbrace{\mathbb{E}_{(y_w, y_l) \sim \mu} [f(\beta \rho_\theta)]}_{\text{offline loss}} \approx f(0) + \underbrace{f'(0)\beta \cdot \mathbb{E}_{(y_w, y_l) \sim \mu} [\rho_\theta]}_{\text{preference optimization}} + \underbrace{\frac{f''(0)\beta^2}{2} \cdot \mathbb{E}_{(y_w, y_l) \sim \mu} [\rho_\theta^2]}_{\text{offline regularization}},$$



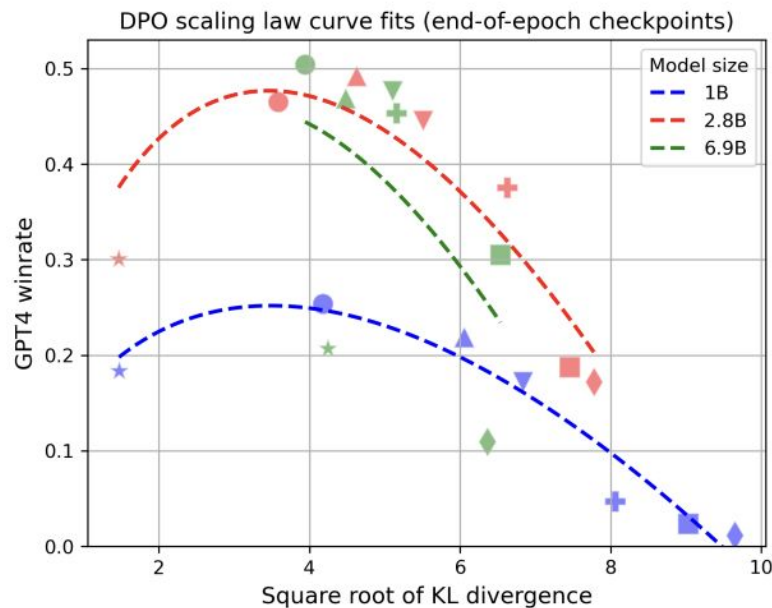
Over-optimization in DPO

Like regular RLHF, DPO cannot escape from over-optimization with a finite dataset

Over-optimization in DPO

Like regular RLHF, DPO cannot escape from over-optimization with a finite dataset

- As you train more, the performance first increases then decreases
- Over-optimization happens for both small and big models



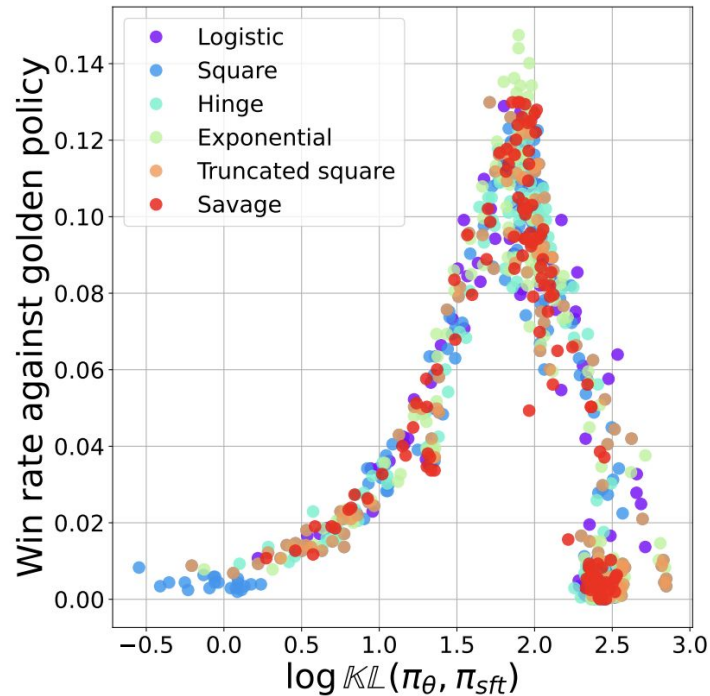
Over-optimization in general offline losses

All loss variants are equivalent in a certain sense

Over-optimization in general offline losses

All loss variants are equivalent in a certain sense

- They all experience over-optimization
- They exhibit a similar trend when calibrating for the KL divergence



Summary

- Making use of the equivalence between reward and policy
 - We can convert reward learning loss into a policy loss
- Advantage of such an approach
 - No more two-stage optimization (reward + policy), just policy optimization
 - Like a SFT loss, there is no sampling, so implementation is much simpler and computation is cheaper
 - Performance gain over SFT is also showcased in certain cases

Online vs. offline RLHF


Turning Offline algorithm into online algorithm

- DPO is a popular form of offline RLHF
 - Not need to sample from the model, just do SFT-like loss
- Reexamine the loss: the sampling distribution μ is fixed

$$\mathbb{E}_{(y_w, y_l) \sim \mu} \left[f \left(\beta \cdot \left(\log \frac{\pi_{\theta}(y_w)}{\pi_{\text{ref}}(y_w)} - \log \frac{\pi_{\theta}(y_l)}{\pi_{\text{ref}}(y_l)} \right) \right) \right]$$

Turning Offline algorithm into online algorithm


- DPO is a popular form of offline RLHF
 - Not need to sample from the model, just do SFT-like loss
- Reexamine the loss: the sampling distribution μ is fixed

$$\mathbb{E}_{(y_w, y_l) \sim \mu} \left[f \left(\beta \cdot \left(\log \frac{\pi_{\theta}(y_w)}{\pi_{\text{ref}}(y_w)} - \log \frac{\pi_{\theta}(y_l)}{\pi_{\text{ref}}(y_l)} \right) \right) \right]$$


Changing the sampling distribution

- If we replace the sampling distribution by the current policy and still calculate the gradient as before


$$\mathbb{E}_{(y_w, y_l) \sim \mu} \left[f \left(\beta \cdot \left(\log \frac{\pi_{\theta}(y_w)}{\pi_{\text{ref}}(y_w)} - \log \frac{\pi_{\theta}(y_l)}{\pi_{\text{ref}}(y_l)} \right) \right) \right]$$

 π_{θ}

Changing the sampling distribution

- If we replace the sampling distribution by the current policy and still calculate the gradient as before

$$\mathbb{E}_{(y_w, y_l) \sim \mu} \left[f \left(\beta \cdot \left(\log \frac{\pi_{\theta}(y_w)}{\pi_{\text{ref}}(y_w)} - \log \frac{\pi_{\theta}(y_l)}{\pi_{\text{ref}}(y_l)} \right) \right) \right]$$

 π_{θ}

- We can show that the gradient looks a lot like the gradient of the RLHF problem

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

Online vs. offline algorithm is parameterized by sampling

- Turned out that we can simply think of online vs. offline RLHF as being parameterized by the sampling distribution

$$\mathbb{E}_{(y_w, y_l) \sim \mu} \left[f \left(\beta \cdot \left(\log \frac{\pi_{\theta}(y_w)}{\pi_{\text{ref}}(y_w)} - \log \frac{\pi_{\theta}(y_l)}{\pi_{\text{ref}}(y_l)} \right) \right) \right]$$

- When sampling is static \rightarrow Offline (DPO, IPO, SLiC...)
- When sampling is dynamic \rightarrow Online (online DPO / IPO...)

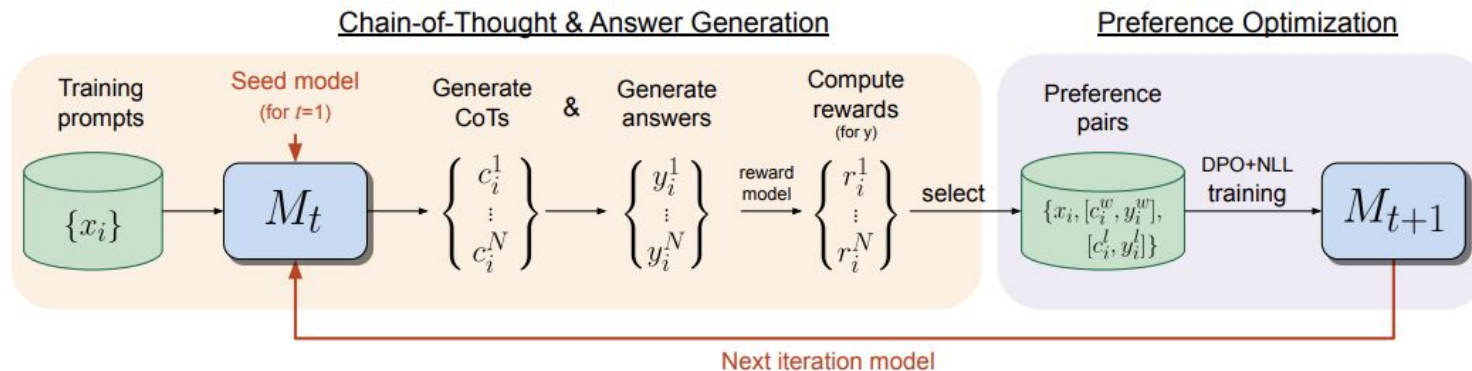
A continuous spectrum

- A special case in the middle: iterative DPO

**DPO, IPO,
SLiC...**

Iterative DPO

**Online RLHF
(DPO, IPO...)**



Why is online needed at all if we have offline?

- Offline is superior in terms of compute efficiency vs. online
- Offline is theoretical equivalent to online
 - Recall the theoretical derivation
- Can we just not learn the optimal policy from offline data?
 - No, since theory breaks easily in practice :)

Lessons from the RL literature

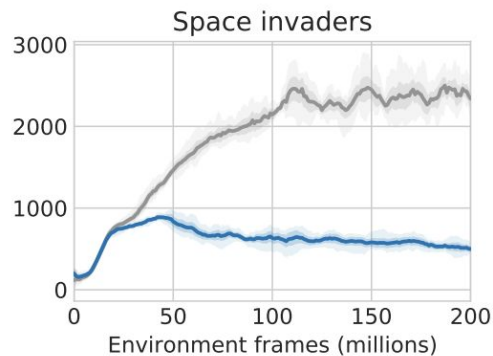
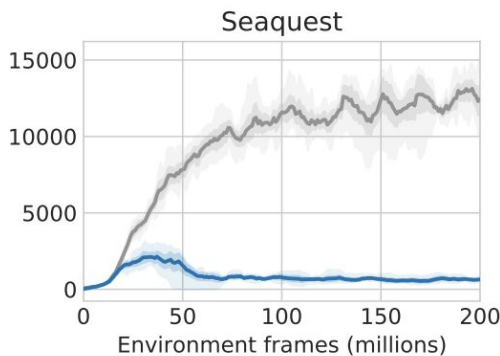
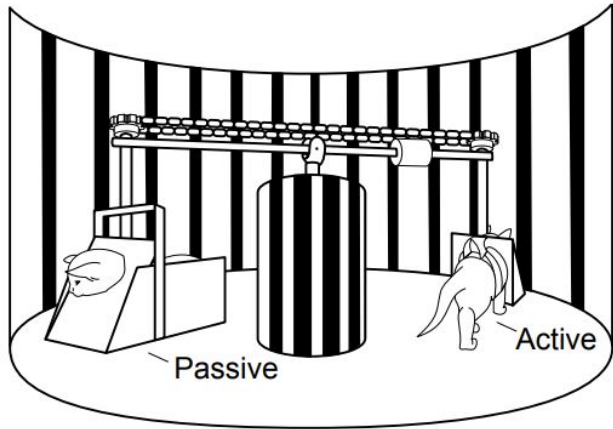
Tandem effect

- Learning from offline data is challenging in general
- Online interactive data collection is key to maintaining stable learning

Lessons from the RL literature

Tandem effect

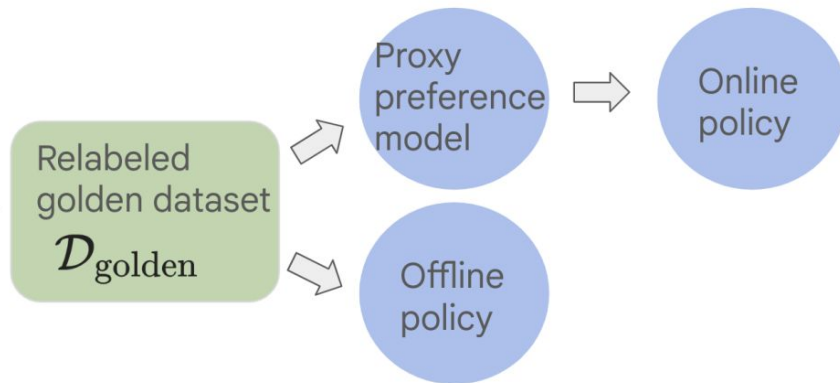
- Learning from offline data is challenging in general
- Online interactive data collection is key to maintaining stable learning



Online vs. offline learning gap in RLHF

For a controlled study, we start with the same preference data, we proceed in two ways of learning

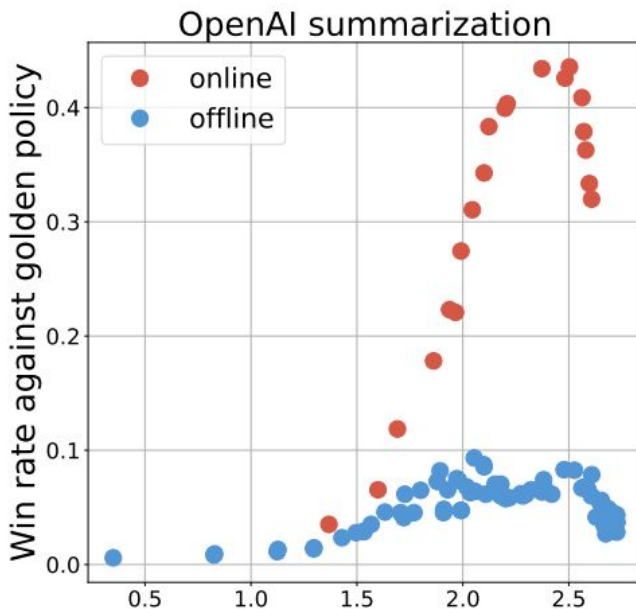
- Online RLHF: reward model + online policy optimization
- Offline RLHF: direct policy optimization with offline data



Online vs. offline learning gap in RLHF

After training, we evaluate with “golden” reward

- Both algorithms suffer from over-optimization at some point
- Online leads to better performance overall compared to offline
- Observations hold for multiple datasets and training hyper-parameters



Challenges and trade-offs

- Online and offline learning is continuously parameterized
 - In practice, it is typically somewhere in the middle

Challenges and trade-offs

- Online and offline learning is continuously parameterized
 - In practice, it is typically somewhere in the middle
- Even with static dataset, it is still beneficial to do “online RLHF”
 - Hypothesis: reward model is a discriminative task, and is easier than direct generative task
 - Observation: online learning is just more stable than offline learning, even with imperfect rewards
- Lesson: always good to collect offline data in a semi-online fashion
 - Help identify the “right” missing data

Other topics in RLHF

Not just human feedback

Human feedback is the canonical defining property of RLHF, other feedback is also available and valuable

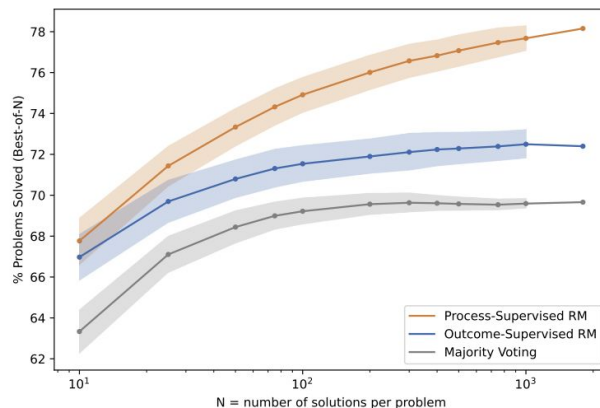
- Model's self-assessment, model as a judge
- Ground truth solution feedback, available in reasoning, coding and more “objective” domains

Not just feedback at the end of a generation

Thus far human feedback is at the end of a generation

- We might get richer feedback if we ask human raters to rate the middle part of a generation
- Example: check if a solution is wrong in the middle
- More fine-grained feedback → Better RM

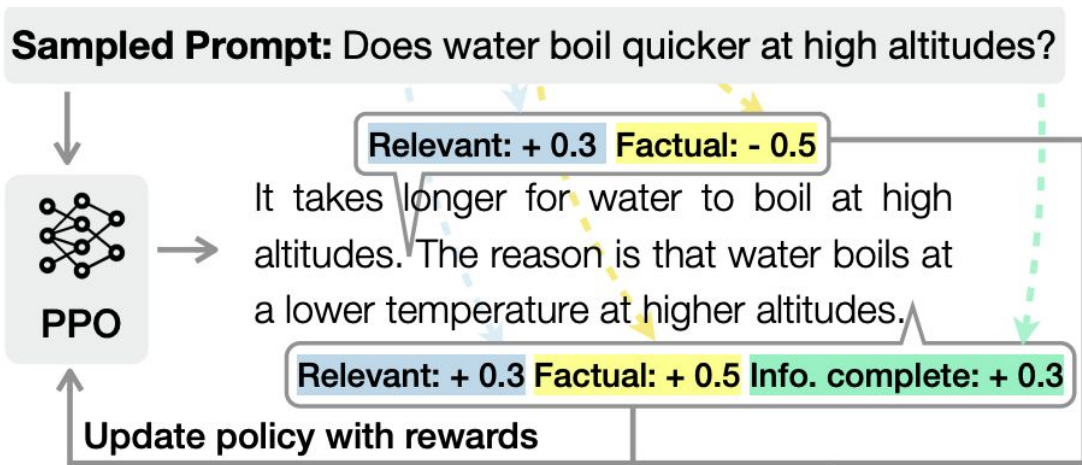
	ORM	PRM	Majority Voting
% Solved (Best-of-1860)	72.4	78.2	69.6



More advanced reward models

We can also model reward in a way that provides more signals to the policy optimization process

- Inter-generation reward
 - Is this sentence related?
- Multiple reward
 - Factuality?
 - Relevance?
 - Completeness?



Nash equilibrium as a solution concept

Lots of assumptions thus far

- Bradley-Terry model
- Reward maximization

All these point to the assumption that all generations can be ranked monotonically.
What if we have “rock, paper, scissor” kind of situation with generations?

Nash equilibrium as a solution concept

Reward maximization is only applicable if all responses can be ranked

- Instead of a reward model, we should have learned a preference model
 - $\mathbf{r}(\mathbf{x}, \mathbf{y1}, \mathbf{y2})$ which is not necessarily representable as rewards
- Solution concept: Nash equilibrium instead of maximization

$$\pi^* \stackrel{\text{def}}{=} \arg \max_{\pi} \min_{\pi'} \mathcal{P}(\pi \succ \pi')$$

Summary of the whole presentation

- RLHF is built on RL
 - Differ from the fact that does not have a “perfect reward”
 - Human feedback needs to be modeled as RM
- RLHF optimizes against a learned RM
 - To bypass the trouble we investigate DPO and its generalized variants
 - DPO is more compute efficient but learns only with offline data
- RLHF benefits from online data collection in general
- So many open problems in RLHF!

Thank you!