Title: Reinforcement learning from human feedback

URL: https://en.wikipedia.org/wiki/Reinforcement_learning_from_human_feedback

PageID: 73200355

Categories: Category:2017 in artificial intelligence, Category:Language modeling, Category:Reinforcement learning

---

Supervised learning

Unsupervised learning

Semi-supervised learning

Self-supervised learning

Reinforcement learning

Meta-learning

Online learning

Batch learning

Curriculum learning

Rule-based learning

Neuro-symbolic AI

Neuromorphic engineering

Quantum machine learning

Classification

Generative modeling

Regression

Clustering

Dimensionality reduction

Density estimation

Anomaly detection

Data cleaning

AutoML

Association rules

Semantic analysis

Structured prediction

Feature engineering

Feature learning

Learning to rank

Grammar induction

Ontology learning

Multimodal learning

Apprenticeship learning

Decision trees

Ensembles Bagging Boosting Random forest

Bagging

Boosting

Random forest

k -NN

Linear regression

Naive Bayes

Artificial neural networks

Logistic regression

Perceptron

Relevance vector machine (RVM)

Support vector machine (SVM)

BIRCH

CURE

Hierarchical

k -means

Fuzzy

Expectation–maximization (EM)

DBSCAN

OPTICS

Mean shift

Factor analysis

CCA

ICA

LDA

NMF

PCA

PGD

t-SNE

SDL

Graphical models Bayes net Conditional random field Hidden Markov

Bayes net

Conditional random field

Hidden Markov

RANSAC

k -NN

Local outlier factor

Isolation forest

Autoencoder

Deep learning

Feedforward neural network

Recurrent neural network LSTM GRU ESN reservoir computing

LSTM

GRU

ESN

reservoir computing

Boltzmann machine Restricted

Restricted

GAN

Diffusion model

SOM

Convolutional neural network U-Net LeNet AlexNet DeepDream

U-Net

LeNet

AlexNet

DeepDream

Neural field Neural radiance field Physics-informed neural networks

Neural radiance field

Physics-informed neural networks

Transformer Vision

Vision

Mamba

Spiking neural network

Memtransistor

Electrochemical RAM (ECRAM)

Q-learning

Policy gradient

SARSA

Temporal difference (TD)

Multi-agent Self-play

Self-play

Active learning

Crowdsourcing

Human-in-the-loop

Mechanistic interpretability

RLHF

Coefficient of determination

Confusion matrix

Learning curve

ROC curve

Kernel machines

Bias–variance tradeoff

Computational learning theory

Empirical risk minimization

Occam learning

PAC learning

Statistical learning

VC theory

Topological deep learning

AAAI

ECML PKDD

NeurIPS

ICML

ICLR

IJCAI

ML

JMLR

Glossary of artificial intelligence

List of datasets for machine-learning research List of datasets in computer vision and image processing

List of datasets in computer vision and image processing

Outline of machine learning

v

t

e

In machine learning , reinforcement learning from human feedback (RLHF) is a technique to align an intelligent agent with human preferences. It involves training a reward model to represent preferences, which can then be used to train other models through reinforcement learning .

In classical reinforcement learning, an intelligent agent's goal is to learn a function that guides its behavior, called a policy . This function is iteratively updated to maximize rewards based on the agent's task performance. [1] However, explicitly defining a reward function that accurately approximates human preferences is challenging. Therefore, RLHF seeks to train a "reward model" directly from human feedback . [2] The reward model is first trained in a supervised manner to predict if a response to a given prompt is good (high reward) or bad (low reward) based on ranking data collected from human annotators . This model then serves as a reward function to improve an

agent's policy through an optimization algorithm like proximal policy optimization . [3] [4] [5]

RLHF has applications in various domains in machine learning, including natural language processing tasks such as text summarization and conversational agents , computer vision tasks like text-to-image models , and the development of video game bots . While RLHF is an effective method of training models to act better in accordance with human preferences, it also faces challenges due to the way the human preference data is collected. Though RLHF does not require massive amounts of data to improve performance, sourcing high-quality preference data is still an expensive process. Furthermore, if the data is not carefully collected from a representative sample , the resulting model may exhibit unwanted biases .

Background and motivation

Optimizing a model based on human feedback is desirable when a task is difficult to specify yet easy to judge. [6] For example, one may want to train a model to generate safe text that is both helpful and harmless (such as lacking bias , toxicity, or otherwise harmful content). Asking humans to manually create examples of harmless and harmful text would be difficult and time-consuming. However, humans are adept at swiftly assessing and comparing the harmfulness of different AI-generated text. Therefore, a more practical objective would be to allow the model to use this type of human feedback to improve its text generation. [7]

Despite the clear benefits of incorporating human feedback in training models, prior efforts—including some that leverage reinforcement learning —have encountered significant challenges. Most attempts were either narrow and difficult to generalize, breaking down on more complex tasks, [8] [9] [10] [11] or they faced difficulties learning from sparse (lacking specific information and relating to large amounts of text at a time) or noisy (inconsistently rewarding similar outputs) reward functions. [12] [13]

RLHF was not the first successful method of using human feedback for reinforcement learning, but it is one of the most widely used. The foundation for RLHF was introduced as an attempt to create a general algorithm for learning from a practical amount of human feedback. [6] [3] The algorithm as used today was introduced by OpenAI in a paper on enhancing text continuation or summarization based on human feedback, and it began to gain popularity when the same method was reused in their paper on InstructGPT . [2] [14] [15] RLHF has also been shown to improve the robustness of RL agents and their capacity for exploration , which results in an optimization process more adept at handling uncertainty and efficiently exploring its environment in search of the highest reward. [16]

Collecting human feedback

Human feedback is commonly collected by prompting humans to rank instances of the agent's behavior. [15] [17] [18] These rankings can then be used to score outputs, for example, using the Elo rating system , which is an algorithm for calculating the relative skill levels of players in a game based only on the outcome of each game. [3] While ranking outputs is the most widely adopted form of feedback, recent research has explored other forms, such as numerical feedback, natural language feedback, and prompting for direct edits to the model's output. [19]

One initial motivation of RLHF was that it requires relatively small amounts of comparison data to be effective. [6] It has been shown that a small amount of data can lead to comparable results to a larger amount. In addition, increasing the amount of data tends to be less effective than proportionally increasing the size of the reward model. [14] Nevertheless, a larger and more diverse amount of data can be crucial for tasks where it is important to avoid bias from a partially representative group of annotators. [15]

When learning from human feedback through pairwise comparison under the Bradley–Terry–Luce model (or the Plackett–Luce model for K-wise comparisons over more than two comparisons), the maximum likelihood estimator (MLE) for linear reward functions has been shown to converge if the comparison data is generated under a well-specified linear model . This implies that, under certain conditions, if a model is trained to decide which choices people would prefer between pairs (or groups) of choices, it will necessarily improve at predicting future preferences. This improvement is expected as long as the comparisons it learns from are based on a consistent and simple rule. [20]

[21]

Both offline data collection models, where the model is learning by interacting with a static dataset and updating its policy in batches, as well as online data collection models, where the model directly interacts with the dynamic environment and updates its policy immediately, have been mathematically studied proving sample complexity bounds for RLHF under different feedback models. [20] [22]

In the offline data collection model, when the objective is policy training, a pessimistic MLE that incorporates a lower confidence bound as the reward estimate is most effective. Moreover, when applicable, it has been shown that considering K-wise comparisons directly is asymptotically more efficient than converting them into pairwise comparisons for prediction purposes. [22] [23] [15]

In the online scenario, when human feedback is collected through pairwise comparisons under the Bradley–Terry–Luce model and the objective is to minimize the algorithm's regret (the difference in performance compared to an optimal agent), it has been shown that an optimistic MLE that incorporates an upper confidence bound as the reward estimate can be used to design sample efficient algorithms (meaning that they require relatively little training data). A key challenge in RLHF when learning from pairwise (or dueling) comparisons is associated with the non-Markovian nature of its optimal policies. Unlike simpler scenarios where the optimal strategy does not require memory of past actions, in RLHF, the best course of action often depends on previous events and decisions, making the strategy inherently memory-dependent. [21]

Applications

RLHF has been applied to various domains of natural language processing (NLP), such as conversational agents, text summarization, and natural language understanding. [24] [14] Ordinary reinforcement learning, in which agents learn from their actions based on a predefined "reward function", is difficult to apply to NLP tasks because the rewards tend to be difficult to define or measure, especially when dealing with complex tasks that involve human values or preferences. [6] RLHF can steer NLP models, in particular language models , to provide answers that align with human preferences with regard to such tasks by capturing their preferences beforehand in the reward model. This results in a model capable of generating more relevant responses and rejecting inappropriate or irrelevant queries. [15] [25] Some notable examples of RLHF-trained language models are OpenAI 's ChatGPT (and its predecessor InstructGPT), [17] [26] [27] DeepMind 's Sparrow , [28] [29] [30] Google 's Gemini , [31] and Anthropic 's Claude . [32]

In computer vision, RLHF has also been used to align text-to-image models . Studies that successfully used RLHF for this goal have noted that the use of KL regularization in RLHF, which aims to prevent the learned policy from straying too far from the unaligned model, helped to stabilize the training process by reducing overfitting to the reward model. The final image outputs from models trained with KL regularization were noted to be of significantly higher quality than those trained without. [33] [34] Other methods tried to incorporate the feedback through more direct training—based on maximizing the reward without the use of reinforcement learning—but conceded that an RLHF-based approach would likely perform better due to the online sample generation used in RLHF during updates as well as the aforementioned KL regularization over the prior model, which mitigates overfitting to the reward function. [35]

RLHF was initially applied to other areas, such as the development of video game bots and tasks in simulated robotics . For example, OpenAI and DeepMind trained agents to play Atari games based on human preferences. In classical RL-based training of such bots, the reward function is simply correlated to how well the agent is performing in the game, usually using metrics like the in-game score . In comparison, in RLHF, a human is periodically presented with two clips of the agent's behavior in the game and must decide which one looks better. This approach can teach agents to perform at a competitive level without ever having access to their score. In fact, it was shown that RLHF can sometimes lead to superior performance over RL with score metrics because the human's preferences can contain more useful information than performance-based metrics. [6] [36] The agents achieved strong performance in many of the environments tested, often surpassing human performance. [37]

Training

In RLHF, two different models are trained: a reward model and a reinforcement learning (RL) policy. The reward model learns to determine what behavior is desirable based on human feedback, while the policy is guided by the reward model to determine the agent's actions. Both models are commonly initialized using a pre-trained autoregressive language model . This model is then customarily trained in a supervised manner on a relatively small dataset of pairs of prompts to an assistant and their accompanying responses, written by human annotators.

Reward model

The reward model is usually initialized with a pre-trained model, as this initializes it with an understanding of language and focuses training explicitly on learning human preferences. In addition to being used to initialize the reward model and the RL policy, the model is then also used to sample data to be compared by annotators. [15] [14]

The reward model is then trained by replacing the final layer of the previous model with a randomly initialized regression head. This change shifts the model from its original classification task over its vocabulary to simply outputting a number corresponding to the score of any given prompt and response. This model is trained on the human preference comparison data collected earlier from the supervised model. In particular, it is trained to minimize the following cross-entropy loss function: L (θ) = – 1 (K 2) E (x , y w , y l) [log ■ (σ (r θ (x , y w) – r θ (x , y l)))] = – 1 (K 2) E (x , y w , y l) log ■ [e r θ (x , y w) e r θ (x , y w) + e r θ (x , y l)] {\displaystyle {\mathcal {L}}(\theta)=-{\frac {1}{K \choose 2}}E_{(x,y_{w},y_{l})}[\log(\sigma (r_{\theta }(x,y_{w})-r_{\theta }(x,y_{l})))]=-{\frac {1}{K \choose 2}}E_{(x,y_{w},y_{l})}\log \left[{\frac {e^{r_{\theta }(x,y_{w})}}{e^{r_{\theta }(x,y_{w})}+e^{r_{\theta }(x,y_{l})}}}\right]}

where K {\displaystyle K} is the number of responses the labelers ranked, r θ (x , y) {\displaystyle r_{\theta }(x,y)} is the output of the reward model for prompt x {\displaystyle x} and completion y {\displaystyle y} , y w {\displaystyle y_{w}} is the preferred completion over y l {\displaystyle y_{l}} , σ (x) {\displaystyle \sigma (x)} denotes the sigmoid function , and E [X] {\displaystyle E[X]} denotes the expected value . [15] This can be thought of as a form of logistic regression , where the model predicts the probability that a response y w {\displaystyle y_{w}} is preferred over y l {\displaystyle y_{l}} .

This loss function essentially measures the difference between the reward model's predictions and the decisions made by humans. The goal is to make the model's guesses as close as possible to the humans' preferences by minimizing the difference measured by this equation. In the case of only pairwise comparisons, K = 2 {\displaystyle K=2} , so the factor of 1 / (K 2) = 1 {\displaystyle 1/{\tbinom {K}{2}}=1} . [14] In general, all (K 2) {\displaystyle {\tbinom {K}{2}}} comparisons from each prompt are used for training as a single batch . [15]

After training, the outputs of the model are normalized such that the reference completions have a mean score of 0. That is, [14] ∑ y r θ (x , y) = 0 {\textstyle \sum _{y}r_{\theta }(x,y)=0} for each query and reference pair (x , y) {\displaystyle (x,y)} by calculating the mean reward across the training dataset and setting it as the bias in the reward head.

Policy

Similarly to the reward model, the human feedback policy is also initialized from a pre-trained model. [14]

The key is to understand language generation as if it is a game to be learned by RL. In RL, a policy is a function that maps a game state to a game action. In RLHF, the "game" is the game of replying to prompts. A prompt is a game state, and a response is a game action. This is a fairly trivial kind of game, since every game lasts for exactly one step. Nevertheless, it is a game, and so RL algorithms can be applied to it.

The first step in its training is supervised fine-tuning (SFT). This step does not require the reward model. Instead, the pre-trained model is trained on a dataset D S F T {\displaystyle D_{SFT}} that contains prompt-response pairs (x , y) {\displaystyle (x,y)} . Then, during SFT, the model is trained to auto-regressively generate the corresponding response y {\displaystyle y} when given a random prompt x {\displaystyle x} . The original paper recommends to SFT for only one epoch, since more

than that causes overfitting.

The dataset D S F T {\displaystyle D_{SFT}} is usually written by human contractors, who write both the prompts and responses.

The second step uses a policy gradient method to the reward model. It uses a dataset D R L {\displaystyle D_{RL}} , which contains prompts, but not responses. Like most policy gradient methods, this algorithm has an outer loop and two inner loops:

Initialize the policy π φ R L {\displaystyle \pi _{\phi }^{RL}} to π S F T {\displaystyle \pi ^{SFT}} , the policy output from SFT.

Loop for many steps. Initialize a new empty dataset D π φ R L {\displaystyle D_{\pi _{\phi }^{RL}}} . Loop for many steps Sample a random prompt x {\displaystyle x} from D R L {\displaystyle D_{RL}} . Generate a response y {\displaystyle y} from the policy π φ R L {\displaystyle \pi _{\phi }^{RL}} . Calculate the reward signal r θ (x , y) {\displaystyle r_{\theta }(x,y)} from the reward model r θ {\displaystyle r_{\theta }} . Add the triple (x , y , r θ (x , y)) {\displaystyle (x,y,r_{\theta }(x,y))} to D π φ R L {\displaystyle D_{\pi _{\phi }^{RL}}} . Update φ {\displaystyle \phi } by a policy gradient method to increase the objective function objective (φ) = E (x , y) ~ D π φ RL [r θ (x , y) – β log ■ (π φ RL (y | x) π SFT (y | x))] {\displaystyle {\text{objective}}(\phi)=E_{(x,y)\sim D_{\pi _{\phi }^{\text{RL}}}}\left[r_{\theta }(x,y)-\beta \log \left({\frac {\pi _{\phi }^{\text{RL}}(y|x)}{\pi ^{\text{SFT}}(y|x)}}\right)\right]}

Initialize a new empty dataset D π φ R L {\displaystyle D_{\pi _{\phi }^{RL}}} .

Loop for many steps Sample a random prompt x {\displaystyle x} from D R L {\displaystyle D_{RL}} . Generate a response y {\displaystyle y} from the policy π φ R L {\displaystyle \pi _{\phi }^{RL}} . Calculate the reward signal r θ (x , y) {\displaystyle r_{\theta }(x,y)} from the reward model r θ {\displaystyle r_{\theta }} . Add the triple (x , y , r θ (x , y)) {\displaystyle (x,y,r_{\theta }(x,y))} to D π φ R L {\displaystyle D_{\pi _{\phi }^{RL}}} .

Sample a random prompt x {\displaystyle x} from D R L {\displaystyle D_{RL}} .

Generate a response y {\displaystyle y} from the policy π φ R L {\displaystyle \pi _{\phi }^{RL}} .

Calculate the reward signal r θ (x , y) {\displaystyle r_{\theta }(x,y)} from the reward model r θ {\displaystyle r_{\theta }} .

Add the triple (x , y , r θ (x , y)) {\displaystyle (x,y,r_{\theta }(x,y))} to D π φ R L {\displaystyle D_{\pi _{\phi }^{RL}}} .

Update φ {\displaystyle \phi } by a policy gradient method to increase the objective function objective (φ) = E (x , y) ~ D π φ RL [r θ (x , y) – β log ■ (π φ RL (y | x) π SFT (y | x))] {\displaystyle {\text{objective}}(\phi)=E_{(x,y)\sim D_{\pi _{\phi }^{\text{RL}}}}\left[r_{\theta }(x,y)-\beta \log \left({\frac {\pi _{\phi }^{\text{RL}}(y|x)}{\pi ^{\text{SFT}}(y|x)}}\right)\right]}

Note that (x , y) ~ D π φ RL {\displaystyle (x,y)\sim D_{\pi _{\phi }^{\text{RL}}}} is equivalent to x ~ D R L , y ~ π φ RL (· | x) {\displaystyle x\sim D_{RL},y\sim \pi _{\phi }^{\text{RL}}(\cdot |x)} , which means "sample a prompt from D R L {\displaystyle D_{RL}} , then sample a response from the policy".

The objective function has two parts. The first part is simply the expected reward E [r] {\displaystyle E[r]} , and is standard for any RL algorithm. The second part is a "penalty term" involving the KL divergence . The strength of the penalty term is determined by the hyperparameter β {\displaystyle \beta } .

This KL term works by penalizing the KL divergence (a measure of statistical distance between distributions) between the model being fine-tuned and the initial supervised model. By choosing an appropriate β {\displaystyle \beta } , the training can balance learning from new data while retaining useful information from the initial model, increasing generalization by avoiding fitting too closely to the new data. Aside from preventing the new model from producing outputs too dissimilar those of the initial model, a second motivation of including the KL term is to encourage the model to output high- entropy text, so as to prevent the model from collapsing to a small number of canned responses . [14]

In simpler terms, the objective function calculates how well the policy's responses are expected to align with human feedback. The policy generates responses to prompts, and each response is evaluated both on how well it matches human preferences (as measured by the reward model) and how similar it is to responses the model would naturally generate. The goal is to balance improving alignment with human preferences while ensuring the model's responses remain diverse and not too far removed from what it has learned during its initial training. This helps the model not only to provide answers that people find useful or agreeable but also to maintain a broad understanding and avoid overly narrow or repetitive responses.

Proximal policy optimization

The policy function is usually trained by proximal policy optimization (PPO) algorithm. That is, the parameter φ {\displaystyle \phi } is trained by gradient ascent on the clipped surrogate function. [15] [14]

Classically, the PPO algorithm employs generalized advantage estimation , which means that there is an extra value estimator V ξ t (x) {\displaystyle V_{\xi _{t}}(x)} , that updates concurrently with the policy π φ t R L {\displaystyle \pi _{\phi _{t}}^{RL}} during PPO training: π φ t R L , V ξ t , π φ t + 1 R L , V ξ t + 1 , … {\displaystyle \pi _{\phi _{t}}^{RL},V_{\xi _{t}},\pi _{\phi _{t+1}}^{RL},V_{\xi _{t+1}},\dots } . The value estimator is used only during training, and not outside of training.

The PPO uses gradient descent on the following clipped surrogate advantage : L PPO (φ) := E x ~ D RL , y ~ π φ t (y | x) [min (π φ R L (y | x) π φ t R L (y | x) A (x , y) , c l i p (π φ R L (y | x) π φ t R L (y | x) , 1 − ■ , 1 + ■) A (x , y))] {\displaystyle L_{\text{PPO}}(\phi):=E_{x\sim D_{\text{RL}},y\sim \pi _{\phi _{t}}(y|x)}\left[\min \left({\frac {\pi _{\phi }^{RL}(y|x)}{\pi _{\phi _{t}}^{RL}(y|x)}}A(x,y),\mathrm {clip} \left({\frac {\pi _{\phi }^{RL}(y|x)}{\pi _{\phi _{t}}^{RL}(y|x)}},1-\epsilon ,1+\epsilon \right)A(x,y)\right)\right]}

where the advantage term A (x , y) {\displaystyle A(x,y)} is defined as r θ (x , y) − V ξ t (x) {\displaystyle r_{\theta }(x,y)-V_{\xi _{t}}(x)} . That is, the advantage is computed as the difference between the reward (the expected return) and the value estimation (the expected return from the policy). This is used to train the policy by gradient ascent on it, usually using a standard momentum-gradient optimizer, like the Adam optimizer .

The original paper initialized the value estimator from the trained reward model. [14] Since PPO is an actor-critic algorithm, the value estimator is updated concurrently with the policy, via minimizing the squared TD-error, which in this case equals the squared advantage term: L TD (ξ) = E (x , y) ~ D π φ t RL [(r θ (x , y) − β log ■ (π φ t RL (y | x) π SFT (y | x)) − V ξ (x)) 2] {\displaystyle L_{\text{TD}}(\xi)=\mathbb {E} _{(x,y)\sim D{\pi _{\phi _{t}}^{\text{RL}}}}\left[\left(r_{\theta }(x,y)-\beta \log \left({\frac {\pi _{\phi _{t}}^{\text{RL}}(y|x)}{\pi ^{\text{SFT}}(y|x)}}\right)-V_{\xi }(x)\right)^{2}\right]} which is minimized by gradient descent on it. Other methods than squared TD-error might be used. See the actor-critic algorithm page for details.

Mixing pretraining gradients

A third term is commonly added to the objective function to prevent the model from catastrophic forgetting. For example, if the model is only trained in customer service, then it might forget general knowledge in geography. To prevent this, the RLHF process incorporates the original language modeling objective. That is, some random texts x {\displaystyle x} are sampled from the original pretraining dataset D pretrain {\displaystyle D_{\text{pretrain}}} , and the model is trained to maximize the log-likelihood of the text log ■ (π φ R L (x)) {\displaystyle \log(\pi _{\phi }^{RL}(x))} . The final objective function is written as:

L (φ) = E (x , y) ~ D π φ RL [r θ (x , y) − β log ■ (π φ RL (y | x) π SFT (y | x))] + γ E x ~ D pretrain [log ■ (π φ RL (x))] {\displaystyle L(\phi)=E_{(x,y)\sim D_{\pi _{\phi }^{\text{RL}}}}\left[r_{\theta }(x,y)-\beta \log \left({\frac {\pi _{\phi }^{\text{RL}}(y|x)}{\pi ^{\text{SFT}}(y|x)}}\right)\right]+\gamma E_{x\sim D_{\text{pretrain}}}[\log(\pi _{\phi }^{\text{RL}}(x))]}

where γ {\displaystyle \gamma } controls the strength of this pretraining term. [15] This combined objective function is called PPO-ptx, where "ptx" means "Mixing Pretraining Gradients". [7] It was first used in the InstructGPT paper. [15]

In total, this objective function defines the method for adjusting the RL policy, blending the aim of aligning with human feedback and maintaining the model's original language understanding.

So, writing out fully explicitly, the PPO-ptx objective function is:

L PPO-ptx (φ) := E (x , y) ~ D π φ t RL [min (π φ R L (y | x) π φ t R L (y | x) A (x , y) , c l i p (π φ R L (y | x) π φ t R L (y | x) , 1 – ■ , 1 + ■) A (x , y)) – β log ■ (π φ RL (y | x) π SFT (y | x))] + γ E x ~ D pretrain [log ■ (π φ RL (x))] {\displaystyle {\begin{aligned}L_{\text{PPO-ptx}}(\phi)&:=E_{(x,y)\sim D_{\pi _{\phi _{t}}^{\text{RL}}}}\left[\min \left({\frac {\pi _{\phi }^{RL}(y|x)}{\pi _{\phi _{t}}^{RL}(y|x)}}A(x,y),\mathrm {clip} \left({\frac {\pi _{\phi }^{RL}(y|x)}{\pi _{\phi _{t}}^{RL}(y|x)}},1-\epsilon ,1+\epsilon \right)A(x,y)\right)-\beta \log \left({\frac {\pi _{\phi }^{\text{RL}}(y|x)}{\pi ^{\text{SFT}}(y|x)}}\right)\right]\\&+\gamma E_{x\sim D_{\text{pretrain}}}[\log(\pi _{\phi }^{\text{RL}}(x))]\end{aligned}}} which is optimized by gradient ascent on it.

Limitations

RLHF suffers from challenges with collecting human feedback, learning a reward model, and optimizing the policy. [38] Compared to data collection for techniques like unsupervised or self-supervised learning , collecting data for RLHF is less scalable and more expensive. Its quality and consistency may vary depending on the task, interface, and the preferences and biases of individual humans. [15] [39]

The effectiveness of RLHF depends on the quality of human feedback. For instance, the model may become biased , favoring certain groups over others, if the feedback lacks impartiality, is inconsistent, or is incorrect. [3] [40] There is a risk of overfitting , where the model memorizes specific feedback examples instead of learning to generalize . For instance, feedback predominantly from a specific demographic might lead the model to learn peculiarities or noise, along with the intended alignment. Excessive alignment to the specific feedback it received (that is, to the bias therein) can lead to the model performing sub-optimally in new contexts or when used by different groups. [41] A single reward function cannot always represent the opinions of diverse groups of people. Even with a representative sample, conflicting views and preferences may result in the reward model favoring the majority's opinion, potentially disadvantaging underrepresented groups. [38]

In some cases, as is possible in regular reinforcement learning , there may be a risk of the model learning to manipulate the feedback process or game the system to achieve higher rewards rather than genuinely improving its performance. [42] In the case of RLHF, a model may learn to exploit the fact that it is rewarded for what is evaluated positively and not necessarily for what is actually good, which can lead to it learning to persuade and manipulate. For example, models might learn that apparent confidence, even if inaccurate, garners higher rewards. Such behavior, if unchecked, is not just incentivized but can cause significant deployment issues due to the model's potential to mislead. Studies have found that humans are not skilled at identifying mistakes in LLM outputs in complex tasks; therefore, models learning to generate confident-sounding yet incorrect text can lead to significant issues when deployed. [38]

Alternatives

Reinforcement learning from AI feedback

Similarly to RLHF, reinforcement learning from AI feedback (RLAIF) relies on training a preference model, except that the feedback is automatically generated. [43] This is notably used in Anthropic 's constitutional AI , where the AI feedback is based on the conformance to the principles of a constitution. [44]

Direct alignment algorithms

Direct alignment algorithms (DAA) have been proposed as a new class of algorithms [45] [46] that seek to directly optimize large language models (LLMs) on human feedback data in a supervised manner instead of the traditional policy-gradient methods.

These algorithms aim to align models with human intent more transparently by removing the intermediate step of training a separate reward model. Instead of first predicting human preferences and then optimizing against those predictions, direct alignment methods train models end-to-end on

human-labeled or curated outputs. This reduces potential misalignment risks introduced by proxy objectives or reward hacking.

By directly optimizing for the behavior preferred by humans, these approaches often enable tighter alignment with human values, improved interpretability , and simpler training pipelines compared to RLHF.

Direct preference optimization

Direct preference optimization (DPO) is a technique to learn human preferences. Like RLHF, it has been applied to align pre-trained large language models using human-generated preference data. Unlike RLHF, however, which first trains a separate intermediate model to understand what good outcomes look like and then teaches the main model how to achieve those outcomes, DPO simplifies the process by directly adjusting the main model according to people's preferences. It uses a change of variables to define the "preference loss " directly as a function of the policy and uses this loss to fine-tune the model, helping it understand and prioritize human preferences without needing a separate step. Essentially, this approach directly shapes the model's decisions based on positive or negative human feedback.

Recall, the pipeline of RLHF is as follows:

We begin by gathering human preference dataset D {\displaystyle D} .

We then fit a reward model r ∗ {\displaystyle r^{*}} to data, by maximum likelihood estimation using the Plackett–Luce model r ∗ = arg ⁡ max ⁡ r E (x , y 1 , … , y N) ∼ D [ln ⁡ ∏ k = 1 N e r (x , y k) ∑ i = k N e r (x , y i)] {\displaystyle r^{*}=\arg \max _{r}\mathbb {E} _{(x,y_{1},\dots ,y_{N})\sim D}\left[\ln \prod _{k=1}^{N}{\frac {e^{r(x,y_{k})}}{\sum _{i=k}^{N}e^{r(x,y_{i})}}}\right]}

We finally train an optimal policy π ∗ {\displaystyle \pi ^{*}} that maximizes the objective function: π ∗ = arg ⁡ max ⁡ π RL E (x , y) ∼ D π RL [r ∗ (x , y) − β log ⁡ (π RL (y | x) π SFT (y | x))] {\displaystyle \pi ^{*}=\arg \max _{\pi ^{\text{RL}}}\mathbb {E} _{(x,y)\sim D_{\pi ^{\text{RL}}}}\left[r^{*}(x,y)-\beta \log \left({\frac {\pi ^{\text{RL}}(y|x)}{\pi ^{\text{SFT}}(y|x)}}\right)\right]}

However, instead of doing the intermediate step of the reward model, DPO directly optimizes for the final policy.

First, solve directly for the optimal policy, which can be done by Lagrange multipliers , as usual in statistical mechanics : π ∗ (y | x) = π SFT (y | x) exp ⁡ (r ∗ (x , y) / β) Z (x) , {\displaystyle \pi ^{*}(y|x)={\frac {\pi ^{\text{SFT}}(y|x)\exp(r^{*}(x,y)/\beta)}{Z(x)}},}

where Z (x) {\displaystyle Z(x)} is the partition function . This is unfortunately not tractable, since it requires summing over all possible responses : Z (x) = ∑ y π SFT (y | x) exp ⁡ (r ∗ (x , y) / β) = E y ∼ π SFT (⋅ | x) [exp ⁡ (r ∗ (x , y) / β)] {\displaystyle Z(x)=\sum _{y}\pi ^{\text{SFT}}(y|x)\exp(r^{*}(x,y)/\beta)=\mathbb {E} _{y\sim \pi ^{\text{SFT}}(\cdot |x)}[\exp(r^{*}(x,y)/\beta)]}

Next, invert this relationship to express the reward implicitly in terms of the optimal policy: r ∗ (x , y) = β log ⁡ π ∗ (y | x) π SFT (y | x) + β log ⁡ Z (x) . {\displaystyle r^{*}(x,y)=\beta \log {\frac {\pi ^{*}(y|x)}{\pi ^{\text{SFT}}(y|x)}}+\beta \log Z(x).}

Finally, plug it back to the maximum likelihood estimator, we obtain [47] : Appendix A π ∗ = arg ⁡ max ⁡ π E (x , y 1 , … , y N) ∼ D [ln ⁡ ∏ k = 1 N e β log ⁡ π (y k | x) π SFT (y k | x) ∑ i = k N e β log ⁡ π (y i | x) π SFT (y i | x)] {\displaystyle \pi ^{*}=\arg \max _{\pi }\mathbb {E} _{(x,y_{1},\dots ,y_{N})\sim D}\left[\ln \prod _{k=1}^{N}{\frac {e^{\beta \log {\frac {\pi (y_{k}|x)}{\pi ^{\text{SFT}}(y_{k}|x)}}}}{\sum _{i=k}^{N}e^{\beta \log {\frac {\pi (y_{i}|x)}{\pi ^{\text{SFT}}(y_{i}|x)}}}}}\right]}

Usually, DPO is used for modeling human preference in pairwise comparisons, so that N = 2 {\displaystyle N=2} . In that case, we have π ∗ = arg ⁡ max ⁡ π E (x , y w , y l) ∼ D [log ⁡ σ (β log ⁡ π (y w | x) π SFT (y w | x) − β log ⁡ π (y l | x) π SFT (y l | x))] {\displaystyle \pi ^{*}=\arg \max _{\pi }\mathbb {E} _{(x,y_{w},y_{l})\sim D}\left[\log \sigma \left(\beta \log {\frac {\pi (y_{w}|x)}{\pi ^{\text{SFT}}(y_{w}|x)}}-\beta \log {\frac {\pi (y_{l}|x)}{\pi ^{\text{SFT}}(y_{l}|x)}}\right)\right]}

DPO eliminates the need for a separate reward model or reinforcement learning loop, treating alignment as a supervised learning problem over preference data. This is simpler to implement and train than RLHF and has been shown to produce comparable and sometimes superior results. [47] Nevertheless, RLHF has also been shown to beat DPO on some datasets, for example, on benchmarks that attempt to measure truthfulness. Therefore, the choice of method may vary depending on the features of the human preference data and the nature of the task. [48]

Identity preference optimization

Identity preference optimization (IPO) [49] is a modification to the original DPO objective that introduces a regularization term to reduce the chance of overfitting. It remains robust to overtraining by assuming noise in the preference data.

Foremost, IPO first applies a non-linear mapping over the probability distribution of preferences Ψ (q) = log ■ (q / (1 – q)) {\displaystyle \Psi (q)=\log(q/(1-q))} instead of the Bradley-Terry assumption to soften the probability of preferences and smooth the labels. Here, Ψ (q) {\displaystyle \Psi (q)} denotes the Ψ {\displaystyle \Psi } preference objective separate from the policy objective. This helps avoid the overfitting issue of the assumption that pairwise preferences can be substituted for point-wise rewards, which weakens the KL regularization by heavily skewing the preference distribution.

As with DPO, IPO is also formulated as an offline learning objective learned over a human preference dataset D {\displaystyle D} . In particular, the IPO introduces a new objective by applying a mapping Ψ {\displaystyle \Psi } over the preference probability distribution. Practically, Ψ {\displaystyle \Psi } is taken as the identity mapping, which results in IPO. Hence, IPO also directly optimizes for the final policy from the preference dataset and bypasses the reward modeling stage by the following objective:

max π θ E [Ψ (p ∗ (y w ■ y l | x))] – β D K L (π θ | | π ref) {\displaystyle \max _{\pi _{\theta }}\mathbb {E} [\Psi (p^{*}(y_{w}\succ y_{l}|x))]-\beta D_{KL}(\pi _{\theta }||\pi _{\text{ref}})}

where p ∗ (y w ■ y l | x) {\displaystyle p^{*}(y_{w}\succ y_{l}|x)} is preference distribution of the chosen responses y w {\displaystyle y_{w}} over the rejected responses y l {\displaystyle y_{l}} . However, since p ∗ {\displaystyle p^{*}} is not observed directly, we sample from a Bernoulli distribution from the offline preference dataset as: p ∗ (y ■ y ′ | x) = E h [I { h prefers y to y ′ given x }] {\displaystyle p^{*}(y\succ y'|x)=\mathbb {E} _{h}[I\{h{\text{ prefers }}y{\text{ to }}y'{\text{ given }}x\}]}

To solve this objective, IPO minimizes the quadratic loss function:

Minimize E (x , y w , y l) ~ D [(h π (x , y w , y l) – I (y w , y l))] 2 = E x , y w , y l) ~ D [I h π (x , y w , y l) – (1 – I) h π (x , y l , y w) – 1 2 β – 1] 2 = E x , y w , y l ~ D [h π (x , y w , y l) – 1 2 β – 1] 2 {\displaystyle {\begin{aligned}{\text{Minimize }}&\mathbb {E} _{(x,y_{w},y_{l})\sim D}[(h_{\pi }(x,y_{w},y_{l})-I(y_{w},y_{l}))]^{2}\\&=\mathbb {E} _{x,y_{w},y_{l})\sim D}[Ih_{\pi }(x,y_{w},y_{l})-(1-I)h_{\pi }(x,y_{l},y_{w})-{\frac {1}{2}}\beta ^{-1}]^{2}\\&=\mathbb {E} _{x,y_{w},y_{l}\sim D}[h_{\pi }(x,y_{w},y_{l})-{\frac {1}{2}}\beta ^{-1}]^{2}\end{aligned}}}

where h π (x , y w , y l) = log ■ (π θ (y w | x) π ref (y w | x))) – log ■ (π θ (y l | x) π ref (y l | x)) {\displaystyle h_{\pi }(x,y_{w},y_{l})=\log \left({\frac {\pi _{\theta }(y_{w}|x)}{\pi _{\text{ref}}(y_{w}|x)}}\right)-\log \left({\frac {\pi _{\theta }(y_{l}|x)}{\pi _{\text{ref}}(y_{l}|x)}}\right)} and I (y w , y l) {\displaystyle I(y_{w},y_{l})} is a function drawn from the Bernoulli distribution from the preference dataset. Here, I (y , y ′) {\displaystyle I(y,y')} is 1 if y {\displaystyle y} is preferred to y ′ {\displaystyle y'} which happens with probability p ∗ (y ■ y ′) {\displaystyle p^{*}(y\succ y')} , and 0 otherwise. As such, the simplification of the expression directly follows from exploiting the symmetry of y {\displaystyle y} and y ′ {\displaystyle y'} from the Bernoulli such that for each datapoint (y w , y l) i ~ D {\displaystyle (y_{w},y_{l})_{i}\sim D} . In particular this symmetry can be represented as (y , y ′ , I (y , y ′)) = (y w , i , y l , i , 1) {\displaystyle (y,y',I(y,y'))=(y_{w,i},y_{l,i},1)} and (y , y ′ , I (y , y ′)) = (y l , i , y w , i , 0) {\displaystyle (y,y',I(y,y'))=(y_{l,i},y_{w,i},0)} with E y [p y] = 1 2 {\displaystyle \mathbb {E} _{y}[p_{y}]={\frac {1}{2}}} and E [I (y , y ′)] = p y {\displaystyle \mathbb {E} [I(y,y')]=p_{y}} .

In summary, IPO can control the gap between the log-likelihood ratios of the policy model and the reference by always regularizing the solution towards the reference model. It allows learning directly from preferences without a reward modelling stage and without relying on the Bradley-Terry modelisation assumption that assumes that pairwise preferences can be substituted with pointwise rewards. [49] Thus, it avoids overfitting to the preference dataset especially when preferences are near deterministic and the KL term fails.

Kahneman-Tversky optimization

Kahneman-Tversky optimization (KTO) [50] is another direct alignment algorithm drawing from prospect theory to model uncertainty in human decisions that may not maximize the expected value.

In general, KTO seeks to optimize a class of new loss functions proposed as “human-aware losses” (HALO) formulated under prospect theory to model “human values” of a query, response pair (x , y) {\textstyle (x,y)} as v (r θ (x , y) – E Q [r θ (x , y) ′]) {\displaystyle v(r_{\theta }(x,y)-E_{Q}[r_{\theta }(x,y)'])} . A function is defined as a human-aware loss for the value described by the general HALO objective:

f (π θ , π ref) = E x , y ~ D [a x , y v (r θ (x , y) – E y ′ ~ Q [r θ (x , y ′)] ■ reference point)] + C D {\displaystyle f(\pi _{\theta },\pi _{\text{ref}})=\mathbb {E} _{x,y\sim D}[a_{x,y}v{\Bigl (}r_{\theta }(x,y)\;-\;\underbrace {E_{y'\sim Q}[\,r_{\theta }(x,y')\,]} _{\text{reference point}}{\Bigr)}]+C_{D}}

where D {\displaystyle D} is the preference data, C D {\displaystyle C_{D}} is some constant relevant to the dataset, and Q {\displaystyle Q} is some distribution representing the baseline or “reference”. Each training example is attached a label a x , y ∈ { + 1 , – 1 } {\displaystyle a_{x,y}\in \{+1,-1\}} that tells us if the example is desirable (we want to push up its reward) and -1 if it’s undesirable (in order to push down its reward). Unlike previous definitions of the reward, KTO defines r θ (x , y) {\displaystyle r_{\theta }(x,y)} as the “implied reward” taken by the log-likelihood ratio between the policy model and the reference model log ■ (π θ (y | x) π ref (y | x)) {\displaystyle \log \left({\frac {\pi _{\theta }(y|x)}{\pi _{\text{ref}}(y|x)}}\right)} . Here, the value function v {\displaystyle v} is a non-linear (typically concave) function that mimics human loss aversion and risk aversion . As opposed to previous preference optimization algorithms, the motivation of KTO lies in maximizing the utility of model outputs from a human perspective rather than maximizing the likelihood of a “better” label (chosen vs. rejected responses). Hence, it constructs a more relaxed generalization to preference distributions by requiring only a binary feedback signal a x , y {\displaystyle a_{x,y}} instead of explicit preference pairs. For each example (x , y) {\displaystyle (x,y)} in the dataset D {\displaystyle D} , KTO explicitly optimizes the HALO objective as:

π θ ∗ = arg ■ max π θ E (x , y) ~ D [γ y – v (x , y)] {\displaystyle \pi _{\theta }^{*}\;=\;\arg \max _{\pi _{\theta }}\;\;\mathbb {E} _{(x,y)\,\sim \,D}{\Bigl [}\gamma _{y}\;-\;v(x,y){\Bigr]}} , where γ y {\displaystyle \gamma _{y}} is a class-specific constant (e.g., γ y = λ D or λ U {\displaystyle \gamma _{y}=\lambda _{D}{\text{ or }}\lambda _{U}}) controlling how strongly the model should push up good outputs vs. push down bad ones. The value function v (x , y) {\displaystyle v(x,y)} is defined piecewise depending on whether y {\displaystyle y} is desirable (λ D {\displaystyle \lambda _{D}}) or undesirable (λ U {\displaystyle \lambda _{U}}):

v (x , y) = { λ D σ (β (r θ (x , y) – z 0)) , if y ~ y d e s i r a b l e ■ x , λ U σ (β (z 0 – r θ (x , y))) , if y ~ y u n d e s i r a b l e ■ x {\displaystyle v(x,y)\;=\;{\begin{cases}\lambda _{D}\,\sigma \!{\bigl (}\,\beta \,{\bigl (}r_{\theta }(x,y)\;-\;z_{0}{\bigr)}{\bigr)},&\quad {\text{if }}y\sim y_{\mathrm {desirable} \mid x},\\[6pt]\lambda _{U}\,\sigma \!{\bigl (}\,\beta \,{\bigl (}z_{0}\;-\;r_{\theta }(x,y){\bigr)}{\bigr)},&\quad {\text{if }}y\sim y_{\mathrm {undesirable} \mid x}\end{cases}}}

and z 0 = K L (π θ (y ′ ■ x) ■ π r e f (y ′ ■ x)) {\textstyle z_{0}=\mathrm {KL} \!{\Bigl (}\,\pi _{\theta }(y'\mid x)\;{\big \Vert }\;\pi _{\mathrm {ref} }(y'\mid x){\Bigr)}} is a baseline given by the Kullback–Leibler divergence. Here, β {\displaystyle \beta } controls how “risk-averse” the value function is (larger β {\displaystyle \beta } = faster saturation in the logistic function σ {\displaystyle \sigma }). Intuitively, desirable outputs push the model to increase r θ {\displaystyle r_{\theta }} so that r θ – z 0 {\displaystyle r_{\theta }-z_{0}} becomes more positive. Undesirable ones push it in the opposite direction, so the reward is less than the reference. Since many real-world feedback pipelines yield "like/dislike" data more easily than pairwise comparisons, KTO is designed to be

data-cheap and to reflect "loss aversion" more directly by using a straightforward notion of "good vs. bad" at the example level.

See also

Human-in-the-loop

Reward-based selection

References

Further reading

"Deep reinforcement learning from human preferences" . NeurIPS . 2017.

"Training language models to follow instructions with human feedback" . NeurIPS . 2022.

"The N Implementation Details of RLHF with PPO" . huggingface.co . 2023-10-24.

"Proximal Policy Optimization — Spinning Up documentation" . spinningup.openai.com . Retrieved 2025-01-26 .

"The N+ Implementation Details of RLHF with PPO: A Case Study on TL;DR Summarization" . COLM . 2024.

v

t

e

History timeline

timeline

Companies

Projects

Parameter Hyperparameter

Hyperparameter

Loss functions

Regression Bias–variance tradeoff Double descent Overfitting

Bias–variance tradeoff

Double descent

Overfitting

Clustering

Gradient descent SGD Quasi-Newton method Conjugate gradient method

SGD

Quasi-Newton method

Conjugate gradient method

Backpropagation

Attention

Convolution

Normalization Batchnorm

Batchnorm

Activation Softmax Sigmoid Rectifier

Softmax

Sigmoid

Rectifier

Gating

Weight initialization

Regularization

Datasets Augmentation

Augmentation

Prompt engineering

Reinforcement learning Q-learning SARSA Imitation Policy gradient

Q-learning

SARSA

Imitation

Policy gradient

Diffusion

Latent diffusion model

Autoregression

Adversary

RAG

Uncanny valley

RLHF

Self-supervised learning

Reflection

Recursive self-improvement

Hallucination

Word embedding

Vibe coding

Machine learning In-context learning

In-context learning

Artificial neural network Deep learning

Deep learning

Language model Large language model NMT

Large language model

NMT

Reasoning language model

Model Context Protocol

Intelligent agent

Artificial human companion

Humanity's Last Exam

Artificial general intelligence (AGI)

AlexNet

WaveNet

Human image synthesis

HWR

OCR

Computer vision

Speech synthesis 15.ai ElevenLabs

15.ai

ElevenLabs

Speech recognition Whisper

Whisper

Facial recognition

AlphaFold

Text-to-image models Aurora DALL-E Firefly Flux Ideogram Imagen Midjourney Recraft Stable Diffusion

Aurora

DALL-E

Firefly

Flux

Ideogram

Imagen

Midjourney

Recraft

Stable Diffusion

Text-to-video models Dream Machine Runway Gen Hailuo AI Kling Sora Veo

Dream Machine

Runway Gen

Hailuo AI

Kling

Sora

Veo

Music generation Riffusion Suno AI Udio

Riffusion

Suno AI

Udio

Word2vec

Seq2seq

GloVe

BERT

T5

Llama

Chinchilla AI

PaLM

GPT 1 2 3 J ChatGPT 4 4o o1 o3 4.5 4.1 o4-mini 5

1

2

3

J

ChatGPT

4

4o

o1

o3

4.5

4.1

o4-mini

5

Claude

Gemini Gemini (language model) Gemma

Gemini (language model)

Gemma

Grok

LaMDA

BLOOM

DBRX

Project Debater

IBM Watson

IBM Watsonx

Granite

PanGu-Σ

DeepSeek

Qwen

AlphaGo

AlphaZero

OpenAI Five
Self-driving car
MuZero
Action selection AutoGPT
AutoGPT
Robot control
Alan Turing
Warren Sturgis McCulloch
Walter Pitts
John von Neumann
Claude Shannon
Shun'ichi Amari
Kunihiko Fukushima
Takeo Kanade
Marvin Minsky
John McCarthy
Nathaniel Rochester
Allen Newell
Cliff Shaw
Herbert A. Simon
Oliver Selfridge
Frank Rosenblatt
Bernard Widrow
Joseph Weizenbaum
Seymour Papert
Seppo Linnainmaa
Paul Werbos
Geoffrey Hinton
John Hopfield
Jürgen Schmidhuber
Yann LeCun
Yoshua Bengio
Lotfi A. Zadeh
Stephen Grossberg
Alex Graves
James Goodnight
Andrew Ng
Fei-Fei Li

Alex Krizhevsky

Ilya Sutskever

Oriol Vinyals

Quoc V. Le

Ian Goodfellow

Demis Hassabis

David Silver

Andrej Karpathy

Ashish Vaswani

Noam Shazeer

Aidan Gomez

John Schulman

Mustafa Suleyman

Jan Leike

Daniel Kokotajlo

François Chollet

Neural Turing machine

Differentiable neural computer

Transformer Vision transformer (ViT)

Vision transformer (ViT)

Recurrent neural network (RNN)

Long short-term memory (LSTM)

Gated recurrent unit (GRU)

Echo state network

Multilayer perceptron (MLP)

Convolutional neural network (CNN)

Residual neural network (RNN)

Highway network

Mamba

Autoencoder

Variational autoencoder (VAE)

Generative adversarial network (GAN)

Graph neural network (GNN)

Category