Generative Models for NLP

Reinforcement Learning for Human Feedback

Joseph Le Roux

January 9, 2026

Outline

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

DATE: 30/12/25

#+END$_{SRC}$

Topic

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

Recap

So far we have seeen:

1. How word (token) vectors are the basis of text representation;
2. How Language Models can generate texts fluently
3. How Transformers have become the ubiquitous neural architectures for LMs

Today

How we can further train a LM to generate

- not only fluent texts
- but also *useful* texts given a task or a context

Using techniques from Reinforcement learning (1) (4)

LMs for interactions

We can use LMs to *reply* to a request by generating from a prompt (see previous lab session)

A probabilistic model for question answering

Given a prompt x (question, instruction...) we can generate a reply y by sampling the conditional distribution:

$$p(y|x) = \frac{p(x, y)}{p(x)}$$

where x, y is the sequence of x concatenated with y (usually with separator token `<SEP>` in between)

With LMs

So in practice we want to learn to predict sequences x `<SEP>` y where:

- x is fixed and is a typical question, and y is the correct answer

We can use a LM for that, trained with cross-entropy per word as before

Issues

- We do not have the correct answers y^* for all questions
- More imporantly, are all replies different from y^* equally bad?

[illegible]

A Typical Architecture for LM Post-training (1)

Goal: *Align* output with user's expectation

1/ Supervised Fine-Tuning

- Next-word prediction on a corpus of texts similar to target texts;
- Usually from human-generated responses (*eg* Text + summary created by humans)
- Model called π_{SFT}

2/ Collect Preference pairs and train an Reward Model

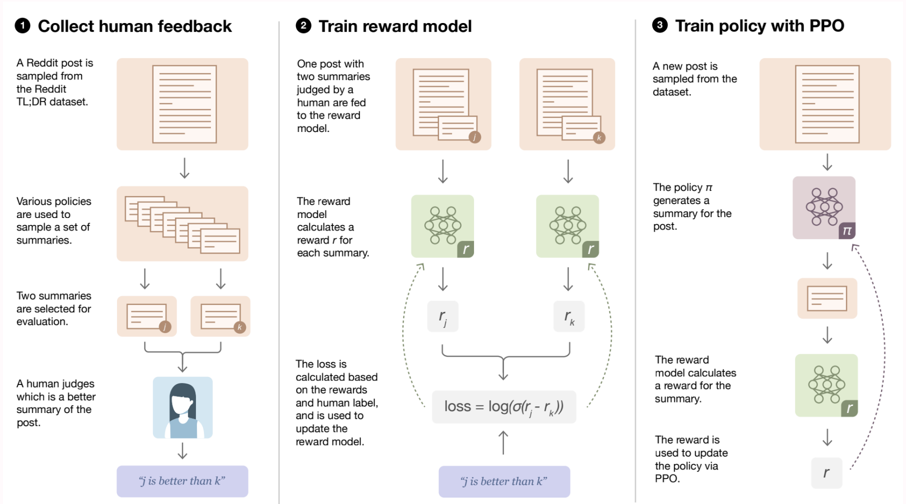
- With SFT (or another model) generate responses $y_1 \dots y_n$ for prompt x
- For each pair of responses,
  1. ask a (human) labeler their preference;
  2. create a corpus of triplets (x, y_c, y_r).
- Train a model R_ϕ to attribute a score to responses to reflect preferences

3/ Train a Policy based on the Reward Model

- initialized as π_{SFT}
- use Reinforcement Learning or Direct Policy Optimization

A Typical Architecture for LM Post-training (2)

Example: Learn to generate summaries (2)



❶ Collect human feedback
A Reddit post is sampled from the Reddit TL;DR dataset.
Various policies are used to sample a set of summaries.
Two summaries are selected for evaluation.
j
k
A human judges which is a better summary of the post.
"j is better than k"
❷ Train reward model
One post with two summaries judged by a human are fed to the reward model.
j
k
The reward model calculates a reward r for each summary.
r
r
r_j
r_k
The loss is calculated based on the rewards and human label, and is used to update the reward model.
loss = $\log(\sigma(r_j - r_k))$
"j is better than k"
❸ Train policy with PPO
A new post is sampled from the dataset.
The policy π generates a summary for the post.
π
The reward model calculates a reward for the summary.
r
The reward is used to update the policy via PPO.
r

Topic

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

Why we need more than just cross-entropy ?

Pros of Cross-entropy Loss

- supervised, self-supervised
- easy to implement
- generates fluent texts (no grammatical errors)
- trained to generate one correct solution

Cons of Cross-Entropy Loss

- works at the word level, not at the text level
- not possible to grade answers
- not possible to add soft preferences

Why Reinforcement Learning (Limits of cross-entropy)

RLHF is one component of post-training.

Post-training is a more complete set of techniques and best-practices to make language models more useful for downstream tasks

RL

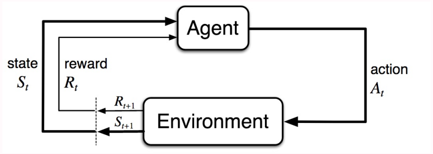
- works at the level of sequences
- grades different replies via a *reward* function
- explore the search space enough to improve the current model

Challenges of RL for text generation

- we do not know the reward function
- we do not want to lose fluency

Reinforcement Learning

Agent-Environnement Model



At each time t:

- the agent witnesses the environment . . .
- . . . which is in state s_t.
- The agent performs an action \$a_t\$. . .
- . . . which transforms the environment to state s_{t+1} and gives reward r_{t+1}, and so on. . .

Definitions

1. The agent will generate trajectories from initial state s_0:
   - $s_0, a_0, r_0, s_1, a_1, r_2, s_2, \ldots r_{T-1} s_T$
2. The function in charge of choosing actions is called the policy π

For LMs

- s_i corresponds to the position i in the reply y

Reinforcement Learning (1)

We want to generate trajectories that earn rewards

- from s_0 (initial state, prompt)
- choose actions (choose words for the reply) from policy π_θ
- so that the sum of all rewards is maximum

A probabilistic variant: stochastic policy

- do not *choose* actions, but rather *sample*
- we need to parametrize a distribution π_θ over actions
- to maximize the *expected sum of rewards*

RL Objective for each example

$$\max_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T-1} r_t]$$

- s_0 corresponds to the prompt for the current example, r_t is the reward received after the t^{th} action (word)

Reinforcement Learning (2)

Probability of a trajectory = Probability of a LM

In our case, the only source of stochasticity is the sampling of each word with policy π_θ

$$\begin{aligned}
p(s_0, a_0, r_1, \ldots, s_T, r_T) &= p(s_0) \times p(a_0, r_0, \ldots, r_{T-1}, s_T | s_0) = p(a_0, r_0, \ldots, r_{T-1}, s_T | s_0) \\
&= p(a_0|s_0) \times p(r_0, \ldots, r_{T-1}, s_T | s_0, a_0) \\
&= \pi_\theta(a_0|s_0) \times p(r_0, \ldots, r_{T-1}, s_T | s_0, a_0) \\
&= \pi_\theta(a_0|s_0) \times p(r_0|s_0, a_0) \times p(s_1, \ldots, r_{T-1}, s_T | s_0, a_0, r_0) \\
&= \pi_\theta(a_0|s_0) \times p(s_0, \ldots, r_{T-1}, s_T | s_0, a_0) \\
&= \pi_\theta(a_0|s_0) \times p(s_0|s_0, a_0) \times p(a_1, \ldots, r_{T-1}, s_T | s_0, a_0, s_1) \\
&= \pi_\theta(a_0|s_0) \times p(a_1, \ldots, r_{T-1}, s_T | s_0, a_0, s_1) \\
&= \pi_\theta(a_0|s_0) \times p(a_1, \ldots, r_{T-1}, s_T | s_1) \\
&= \ldots \\
&= \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t)
\end{aligned}$$

In our case, the probability of a trajectory is the probability $\prod \pi_\theta(y_i|y_{<i}) = p(y)$: a LM !!

Reinforcement Learning (3)

Learn with neural network to parameterize π_θ by gradient descent.

$$
\begin{aligned}
\nabla J(\theta) &= \nabla \mathbb{E}_{\tau\sim\pi}[G(\tau)] \quad \text{where } G(\tau) = \sum_t r_t \text{ in } \tau \\
&= \nabla \sum_\tau p(\tau) G(\tau) \quad \text{(def. expectation)} \\
&= \sum_\tau \nabla p(\tau) G(\tau) \quad \text{(gradient} \leftrightarrow \text{sum)} \\
&= \sum_\tau G(\tau) \nabla p(\tau) \quad \text{(gain is constant)} \\
&= \sum_\tau \frac{p(\tau)}{p(\tau)} G(\tau) \nabla p(\tau) \quad \text{(multiply by one)} \\
&= \sum_\tau p(\tau) G(\tau) \frac{\nabla p(\tau)}{p(\tau)} \quad \text{(rearrange)} \\
&= \sum_\tau p(\tau) G(\tau) \nabla \log p(\tau) \quad \text{(log trick)} \\
\nabla J(\theta) &= \mathbb{E}_{\tau\sim\pi}[G(\tau) \nabla \log p(\tau)] \quad \text{(def. expectation)}
\end{aligned}
$$

Reinforcement Learning (4)

Learn with neural network to parameterize π_θ by gradient descent.

$$\begin{aligned}\nabla J(w) &= \mathbb{E}_{\tau\sim\pi}[G(\tau)\nabla\log(p(\tau))]\\ &= \mathbb{E}_{\tau\sim\pi}[G(\tau)\Big(\sum_{i=0}^{T-1}\nabla\log\pi(a_i|s_i)\Big)]\end{aligned}$$

$\longrightarrow \nabla J(w) \equiv$ Log-likelihood gradient multiplied by G !!

REINFORCE algorithm (5)

While True:

- Sample τ (generate a reply) with the current model with parameters θ
- Compute $G(\tau)$
- Sum log-likelihood losses for all actions in τ multiplied by $G(\tau)$

(can sample multiple τ and average)

Reinforcement Learning (5)

Variance Reduction

- Sampling from the model (MC methods) usually exhibits large variance
- Use a *baseline* that compare $G(\tau)$ with others

REINFORCE with Leave-One-Out Baseline (RLOO)

While True:

- Sample $\tau^1 \dots \tau^K$ (generate K replies) with the current model with parameters θ
- Compute $G(\tau^1) \dots G(\tau^K)$
- Optimize θ with the gradient of:

$$\frac{1}{K}\sum_{k=1}^{K}\big(G(\tau^k) - \frac{1}{K-1}\sum_{k'\neq k} G(\tau^{k'})\big)\big(\sum_{i=0}^{T_k-1}\log\pi(a_i^k|s_i^k)\big)$$

Topic

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

Types of Preferences

Preference Data $\mathcal{D}_{\text{PREF}}$

a collection of triplets (x, y_c, y_r)

- x the prompt (more generally the context);
- y_c the preferred (chosen) the response;
- y_r the rejected response.

y_c is not the best response, simply a better one than y_r

Extensions

Optionally, human labelers can add scores or features to responses. We will ignore this in the following

Learning the Reward Function (1)

Bradley-Terry Model

A BT model of preferences is a model that verifies, for each pair of events i, j:

$$p(i > j) = \frac{p(i)}{p(i) + P(j)}$$

where $i > j$ means that i is preferred to j

Build a BT model from rewards

- Let us define a neural network r_ϕ (LSTM/Transformer...) that given a sequence "x `SEP` y" assign a reward score of y as a response to x;
- We write this score $r_\phi(y)$;
- We can define a probability $p(y) = \frac{\exp r_\phi(y)}{\sum_y' \exp r_\phi(y')}$

We want to maximize that $p(r_\phi(y_c) > r_\phi(y_r)$:

Learning the Reward Function (2)

Build a BT model from rewards

We want to maximize that $p(r_\phi(y_c) > r_\phi(y_r))$:

$$
\begin{aligned}
p(r_\phi(y_c) > r_\phi(y_r)) &= \frac{p(y_c)}{p(y_c) + p(y_r)} \\
&= \frac{\frac{\exp r_\phi(y_c)}{Z}}{\frac{\exp r_\phi(y_c)}{Z} + \frac{\exp r_\phi(y_r)}{Z}} \\
&= \frac{\exp r_\phi(y_c)}{\exp r_\phi(y_c) + \exp r_\phi(y_r)}
\end{aligned}
$$

Learning the Reward Function (3)

Build a BT model from rewards

We want to maximize that $p(r_\phi(y_c) > r_\phi(y_r)$: Equivalently, we want to minimize, by gradient descent:

$$\begin{aligned}
-\log \frac{\exp r_\phi(y_c)}{\exp r_\phi(y_c) + \exp r_\phi(y_r)} &= -\log \frac{1}{1 + \exp(r_\phi(y_r) - r_\phi(y_c))} \\
&= -\log \frac{1}{1 + \exp(r_\phi(y_r) - r_\phi(y_c))} \\
&= -\log 1 + \log(1 + \exp(r_\phi(y_r) - r_\phi(y_c))) \\
L(\phi) &= \log(1 + \exp(r_\phi(y_r) - r_\phi(y_c)))
\end{aligned}$$

Reward Model training

1. Architecture:
   - Usually a simple linear level $h \times 1$ from the CLS/EOS token of the SFT Transformer LM
2. Training
   - Usually just a few epochs (1?)

Topic

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

Policy Learning : Regularization

Issues with the reward model

- usually y_r and y_c are generated by models trained with next-word prediction: very fluent
- the reward does not take fluency into account
- maximizing the expected reward results in non-fluent models

Use Regularization

- we want the final model to be *close* to SFT, so fluency remains.
- use a notion of *close* adapted for distributions: Kullback-Leibler divergence

$$\begin{aligned} D_{KL}(P_{RL}||Q_{SFT}) &= \sum_y P_{RL}(y)\log(\frac{P_{RL}(y)}{Q_{SFT}(y)}) \\ &= \mathbb{E}_{y\sim P_{RL}(\cdot)}[\log P_{RL}(y) - \log Q_{SFT}(y)] \end{aligned}$$

We can approximate this loss by sampling y from the current model.

Policy Learning : Reinforce

REINFORCE with reward from the RM with regularization

Maximize for each example

$$J(\theta) = \frac{1}{K} \sum_{k=1}^{K} R(y^k) \sum_{i=1}^{T^k} \log \pi_\theta(y_i^k | y_{<i}^k)$$

where R is the RLOO reward with RM and regularization:

$$R(y^k) = r_\phi(y^k) - \frac{1}{K-1}(\sum_{k' \neq k} r_\phi(k')) - \lambda_{\mathsf{REG}}(\sum_i (\log \pi_\theta(y_i^k | y_{<i}^k) - \log \pi_{\mathsf{SFT}}(y_i^k | y_{<i}^k)))$$

Policy Learning: Proximal Policy Optimization (PPO) (1)

Another way to implement a policy gradient algorithm:

Define how much an action is better than another one on average $A(s, a)$:

state value $V^\pi(s) = \mathbb{E}[\sum_{k=0}^{T} r_{t+k} | s]$

state-action value $Q^\pi(s, a)\mathbb{E}[\sum_{k=0}^{T} r_{t+k} | s, a]$

advantage $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$

Find new policy with better advantage than previous policy

$$J(\theta) = \frac{1}{T} \sum_i \frac{\pi_\theta(a_i|s_i)}{\pi_{\text{old}}(a_i|s_i)} A^{\pi_{\text{old}}}(s_i, a_i)$$

Policy Learning: Proximal Policy Optimization (PPO) (2)

Issue with PPO: objective very unstable: big changes in θ, difficult to find an optimum

Use a clipped variant (Trust Region Optimization)

$$J^{\text{CLIP}}(\theta) = \frac{1}{T}\sum_i \min[\frac{\pi_\theta(a_i|s_i)}{\pi_{\text{old}}(a_i|s_i)} A^{\pi_{\text{old}}}(s_i, a_i), g(\epsilon, A^{\pi_{\text{old}}}(s_i, a_i))]$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{if } A > 0 \\ (1-\epsilon)A & \text{otherwise.} \end{cases}$$

This means that if $\frac{\pi_\theta(a_i|s_i)}{\pi_{\text{old}}(a_i|s_i)}$ must be close to 1 otherwise the gradient is null and there is no update.

Policy Learning: Proximal Policy Optimization (PPO) (3)

How to compute A in practice?

- $Q(s_i, a_i)$ is approximated by the sum of rewards to go

$$Q(s_i, a_i) = \sum_{t=i}^{T} r_t$$

- $V(s_i)$ is approximated by a neural network v_ϕ
  - typically a linear layer above the Transformer vector of w_i
  - trained with the LM, by mean-squared error

Add entropic regularisation on π_θ

discourage predicting too few actions per state

- $H(\pi_\theta(\cdot|s)) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)$

Final PPO objective:

$$J^{\text{CLIP}}(\theta) + \sum_{i=0}^{T-1} \lambda_1 H(\pi_\theta(\cdot|s_i)) + \frac{\lambda_2}{2}((\sum_{j=i}^{T-1} r_j) - v_\phi(s_i))^2$$

Topic

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

Direct Policy Optimization (1)

Do we need really need reinforcement learning?

- We use RL because we want to incorporate a reward score (not simply 0/1 scores)
- but using full RL with a MDP formulation of LM... is maybe too much?

Can we take into account preferences (x, y_c, y_r) directly?

- *Direct Alignment* algorithms
- Link to the paper (3)

Direct Policy Optimization (2)

Start with the RL Objective with Regularization

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)] - \beta \mathsf{REG}(\theta)$$

- Recall what the probability of a trajectory/response is:
  $\pi_\theta(\tau) = \prod_{(s_i,a_i)\in\tau} \pi_\theta(a_i|s_i)$
- $G(\tau)$ is the sum of rewards for trajectory τ (with possibly RLOO baseline)

RL objective with KL regularization

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)] - \beta\mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)}] = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau) - \beta \log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)}]$$

We would like to see this as a KL divergence between π_θ and ... something easy to compute!

Direct Policy Optimization (3)

DPO as minimizing KL divergence

$$
\begin{aligned}
&\operatorname*{argmax}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau) - \beta \log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)}] \\
=&\operatorname*{argmax}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\frac{1}{\beta}G(\tau) - \log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)}] \\
=&\operatorname*{argmax}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log\exp\frac{1}{\beta}G(\tau) - \log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)}] \\
=&\operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)} - \log\exp\frac{1}{\beta}G(\tau)] \\
=&\operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\pi^{SFT}(\tau)\times\exp\frac{1}{\beta}G(\tau)}] \\
=&\operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\tilde{g}(\tau)}] \qquad \text{with } \tilde{g}(\tau) = \pi^{SFT}(\tau)\times\exp\frac{1}{\beta}G(\tau)
\end{aligned}
$$

Almost there... but $\tilde{g}$ is not a proper distribution (does not sum to 1)

Direct Policy Optimization (4)

From our objective

$$\operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\tilde{g}(\tau)}] \qquad \text{with } \tilde{g}(\tau) = \pi^{SFT}(\tau) \times \exp \frac{1}{\beta} G(\tau)$$

Let us define a normalization for $\tilde{g}$: $z = \sum_{\tau'} \tilde{g}(\tau')$
Note that $g(\tau) = \frac{\tilde{g}(\tau)}{z}$ is a proper distributiuon (positive, sum to one)

We get a KL minimization

$$\begin{aligned}\operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\tilde{g}(\tau)}] &= \operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\tilde{g}(\tau)} + \log z] \\ &= \operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)\times z}{\tilde{g}(\tau)}] \\ &= \operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{\frac{\tilde{g}(\tau)}{z}}] = \operatorname*{argmin}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\log \frac{\pi_\theta(\tau)}{g(\tau)}]\end{aligned}$$

We finally have a KL minimization!

Direct Policy Optimization (5)

What is good about KL minimization

- KL is minimized when the two distribution are equal
- We have the solution of our problem: $\pi_\theta(\tau) = g(\tau)$

But ...

- in practice z (hence r) is not tractable
- G still depends on training a reward model: not very convenient

We can express the sum of reward G from π:

$$\pi_\theta(\tau) = g(\tau) \Leftrightarrow \pi_\theta(\tau) = \frac{\pi^{REF}(\tau) \times \exp \frac{1}{\beta} G(\tau)}{z}$$

$$\Leftrightarrow \frac{\pi_\theta(\tau) \times z}{\pi^{REF}(\tau)} = \exp \frac{1}{\beta} G(\tau)$$

$$\Leftrightarrow \exp \frac{1}{\beta} G(\tau) = \frac{\pi_\theta(\tau) \times z}{\pi^{REF}(\tau)}$$

$$\Leftrightarrow \frac{1}{\beta} G(\tau) = \log \frac{\pi_\theta(\tau) \times z}{\pi^{REF}(\tau)}$$

$$\Leftrightarrow G(\tau) = \beta \log \frac{\pi_\theta(\tau) \times z}{\pi^{REF}(\tau)}$$

$$\Leftrightarrow G(\tau) = \beta \log \frac{\pi_\theta(\tau)}{\pi^{REF}(\tau)} + \log z$$

Solution: From MLE/KL to Contrastive (back to word2vec?)

Recall the preference model probability, and use our definition of G:

$$\begin{aligned}
p(y_c > y_r) &= \frac{\exp G(y_c)}{\exp G(y_c) + \exp G(y_r)} \\
&= \frac{1}{1 + \exp(G(y_r) - G(y_c))} \\
&= \sigma(G(y_c) - G(y_r)) \\
&= \sigma(\beta \log \frac{\pi_\theta(y_c)}{\pi^{REF}(y_c)} + \log z - \beta \log \frac{\pi_\theta(y_r)}{\pi^{REF}(y_r)} - \log z) \\
&= \sigma(\beta \log \frac{\pi_\theta(y_c)}{\pi^{REF}(y_c)} - \beta \log \frac{\pi_\theta(y_r)}{\pi^{REF}(y_r)})
\end{aligned}$$

We can express the probability wtithout explicitly use a reward! We can train from preferences directly

Training

For each triplet (x, y_c, y_r)

- update θ with the gradient of $\mathcal{L}(\theta; x, y_c, y_r) = \log \sigma(\beta \log \frac{\pi_\theta(y_c)}{\pi^{REF}(y_c)} - \beta \log \frac{\pi_\theta(y_r)}{\pi^{REF}(y_r)})$

Topic

- Introduction
- Reinforcement Learning (from Human Feedback) for Generation
- Learning a Reward Function
- Online Policy Learning for Generation
- Offline Policy: DPO
- Conclusion

Bibliography

Nathan Lambert (2024). *Reinforcement Learning from Human Feedback*, Online.
Nisan Stiennon and Long Ouyang and Jeff Wu and Daniel M. Ziegler and Ryan Lowe and Chelsea Voss and Alec Radford and Dario Amodei and Paul F. Christiano (2020). *Learning to summarize from human feedback*, CoRR.
Rafael Rafailov and Archit Sharma and Eric Mitchell and Christopher D Manning and Stefano Ermon and Chelsea Finn (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*.
Sutton, Richard S. and Barto, Andrew G. (2018). *Reinforcement Learning: An Introduction*, The MIT Press.
Williams, R. J. (1992). *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine Learning.