RLHF Mathematics and Intuition in Substantial Detail

Evan Dramko

July 2025

1 Introduction

Reinforcement learning from human feedback (RLHF) has developed into an integral part of the LLM training process. Unlike classic supervised learning which focuses only on accuracy, RLHF allows the engineer to train human preference as an aspect of the model. Due to this, it is often said in literature that RLHF should be used not when training for 'correctness", but rather training for subjective quality of response.

Consider the motivating scenario where a LM has been trained on a massive corpus of data. The data however, reflects information, biases, and response templates that the engineer does not want to be present in the model's response. Instead, the user has a small corpus of sample prompt-answer pairs that they have ranked according to their "goodness". If using supervised learning, a sample prompt-answer pair would need to be generated for nearly every possible unsatisfactory response that the LM could give. Instead, RLHF creates a manner that the engineer's preference can be applied to every potential prompt-answer pair without having to create a sample data point for each specific input.

An important note: there are many variants of RLHF. I am showing the base algorithm here. A common modification is some practitioners prefer to disregard the KL-divergence component and instead clip update sizes. Some new research has moved away from PPO loss in favor of GRPO and other variants.

1.1 Assumed Knowledge

While we cover the mathematics of RLHF in great detail, we do assume that the reader has mathematical maturity, familiarity with deep learning, and is broadly familiar with reinforcement learning (RL). The reader should feel comfortable with the idea of a policy, value function, state-action pairs, etc.

Throughout the sections, we will often denote the mathematics as if the update is being done on a single example at a time for clarity. It can, of course, be batched. To do so, one must adjust the dimensions referenced to account for the addition of the batched sequences.

2 How RLHF Works

We can break down RLHF into five main steps. A detailed analysis of them is provided in the following subsections. Since we are reviewing how RLHF is used right now, we will consider an natural language processing (NLP) use case with a decoder based model. We will denote a prompt as $\mathbf{x}$, a response as $\mathbf{y}$, and will write $\mathbf{xy}$ to denote a prompt-answer pair. In total, the set of all generate prompt and answer pairs is denoted X, Y. Importantly, remember that in natural language processing (NLP) contexts, the state of the system is defined as $\mathbf{xy}$, therefore the MDP the RL process in modeling has an incredibly large number of states.

For clarity, we list the full procedure in Algorithm 1, and detailed analysis of each major step is provided in the following subsections.

Algorithm 1 Batched update step of RLHF

Require: X
Require: $RM(\mathbf{x}, \mathbf{y}) \in \mathbb{R}$
Require: $V(\mathbf{x}) \in \mathbb{R}$
Require: π_{ref}, π_θ
$Y \leftarrow \pi_\theta(X)$
$KL(X, Y) \approx \sum_{(\mathbf{x},\mathbf{y})} \sum_t \left[\ln(\pi_\theta(\mathbf{y}|\mathbf{xy_{i<t}}) - \ln(\pi_{ref}(\mathbf{y}|\mathbf{xy_{i<t}})\right]$
$A(\mathbf{xy}) \leftarrow [RM(\mathbf{xy}) - \beta \cdot KL(\mathbf{xy})] - V(\mathbf{x})$
$c(\mathbf{x}, \mathbf{y}) \leftarrow \exp\left[\ln(\pi_{\theta_j}(\mathbf{y}|\mathbf{x})) - \ln(\pi_{\theta_{j-1}}(\mathbf{y}|\mathbf{x}))\right]$
$\mathcal{L}_{PPO} \leftarrow clip(c(\mathbf{x}, \mathbf{y})) \cdot A(\mathbf{xy})$
$\mathcal{L}_V = MSE\left[V(\mathbf{x}), RM(\mathbf{xy})\right]$
$\mathcal{L}_{expl} = -\sum_t \left[\pi_\theta(\mathbf{y}_t|\mathbf{xy}_{i<t}) \cdot \ln\left(\pi_\theta(\mathbf{y}_t|\mathbf{xy}_{i<t})\right)\right]$
$\mathcal{L} = \mathcal{L}_{PPO} + \alpha_1 \cdot \mathcal{L}_V - \alpha_2 \cdot \mathcal{L}_{expl}$
$backprop(\mathcal{L})$

2.1 Sampling, Scoring, and Reward Model

Prior to beginning RLHF we create a duplicate of our model. We will keep this version static and unchanged. It is referred to as the "reference model", and is used to establish a baseline for model performance. We denote this model as π_{ref}. The other model that we keep active and will continue to update is referred to as the "policy model" and is denoted π_θ.

We draw a sample of prompts from our original dataset, and generate responses to each. We then ask users to rank each response, generating a numeric score from these rankings. These prompts are then used to train a reward model, RM, which outputs a numeric score showing the "goodness" of the response. [1]. We will leave the training of the RM as a different post.

[1] Higher numbers correspond to a better response

2.2 KL-divergence

2.2.1 Reaching the used form

In many RLHF setups, the KL divergence is used to constrain the model such that it will remain "close" to the original reference model. This helps prevent overfitting and unintended "gaming" of the RM. Recall that KL-divergence is defined by:

$$KL_{def}(p||q) = \int p(x) \cdot \frac{\ln(p(\mathbf{x}))}{\ln(q(\mathbf{x}))} \tag{1}$$

$$= \sum_{x \in X} p(x) \cdot \frac{\ln(p(\mathbf{x}))}{\ln(q(\mathbf{x}))} \tag{2}$$

Note that KL-divergence is not symmetric, and thus we use "||" instead of "," in its function notation. In RLHF, we approximate the KL-divergence as:

$$KL(X, Y) \approx \sum_{(\mathbf{x},\mathbf{y})} \sum_{t} [\ln(\pi_\theta(\mathbf{y}|\mathbf{x}\mathbf{y}_{\mathbf{i}<\mathbf{t}}) - \ln(\pi_{ref}(\mathbf{y}|\mathbf{x}\mathbf{y}_{\mathbf{i}<\mathbf{t}})] \tag{3}$$

The careful reader will notice we do not have a $p(\mathbf{x})$ term in our approximation. This term is actually implicit in the approximation. Since we are randomly drawing samples, X, from our dataset, we get an estimate of $p(\mathbf{x})$ just from how often this prompt appears in our fine-tuning data. Also, note that:

$$\frac{\ln(p(\mathbf{x}))}{\ln(q(\mathbf{x}))} = \ln(\pi_\theta(\mathbf{y}|\mathbf{x}) - \ln(\pi_{ref}(\mathbf{y}|\mathbf{x}) \tag{4}$$

We prefer to calculate this term as the LHS of 4 rather than the RHS because we can reuse the component terms later in the algorithm and save the calculations.

2.2.2 How to calculate logprobs

In order to calculate $\ln(\pi(\mathbf{y}|\mathbf{x}\mathbf{y}_{\mathbf{i}<\mathbf{t}})$, when running each forward pass during inference, record the logits over the vocab size. Then, applying a softmax to the logits from each step provides a probability distribution. *Take special care when writing the code: the first token is usually a seed token: "<BOS>" and is fixed. Therefore, it does not need have associated logits since it is never predicted. Additionally, we disregard the logits indexed at the last token value, since the sequence has ended and there is no next token. These logits are undefined.*

2.2.3 Why KL-divergence makes sense

Recall that $\pi(\mathbf{y}|\mathbf{x})$ is the probability that $\mathbf{y}$ occurs from prompt $\mathbf{x}$. It can either be either a continuous or one-hot encoded probability, the mathematics is the same. Notice that if we were to use native probabilities rather than log probabilities, the likelihood of any response would be defined as:

$$\pi_\theta(\mathbf{y}|\mathbf{x}) = \Pi_t \pi(\mathbf{y}_t|\mathbf{x}\mathbf{y}_{\mathbf{i}<\mathbf{t}}) \tag{5}$$
$$\to 0 \text{ as } t \to \infty \tag{6}$$

The use of log probabilities prevents this from happening in practice. While the use of log probs is an artifact of the definition of KL-divergence and was not user-designed specifically for RLHF, it is what allows us to have non-negligible probabilities for our sequences.

2.3 Advantage Calculations

2.3.1 V - Value Function

V refers to the value function commonly seen in RL applications. It returns an estimate for the value of a state as a whole. In this case, when we take $V(\mathbf{x})$, it gives a measure of the average reward we can expect from the responses to the prompt $\mathbf{x}$. The value for V can be hard to know deterministically. It would require calculating every response to a response to every single prompt, and taking their weighted average. Instead, we approximate V through the use of a head trained on the output of the LM. Every time we generate a prompt-response pair, $\mathbf{xy}$, we train V to match $RM(\mathbf{xy})$ [2]. The intuition is that by averaging enough samples, $\sum_i RM(\mathbf{xy_i})$ will approximate $V(\mathbf{x})$ well. In practice, we are unlikely to see any specific $\mathbf{x_i}$ more than once[3], but under function approximation conditions and assumptions (i.e: smoothness of the mapping, sharing of parameters, similarity of distinct $\mathbf{x_i}, \mathbf{x_j}$ etc) with sufficiently large quantities of $\mathbf{x_i}$ we can see useful estimates $V(\mathbf{x_i})$ anyway.

Initially, it can seem confusing that V is trained using jointly with the rest of the network. The most straightforward idea would be to use two separate losses and optimizers: one for the policy and one for V. While this can work, empirical results show training both on a shared loss often produces better results. The intuition is that we want the encoder portion (part prior to the policy and value head) to have a robust shared representation that captures all the meaningful information about the problem. Control over the weighting factor for the loss terms from the policy and value functions is essential to ensure proper training. This approach is not universal however, many prefer to still separate the training of V from the rest of the parameters to prevent "gaming" of the system. As is often the case with RL, there is not a clear-cut procedure... each situation must be considered individually.

2.3.2 Total Advantage

We calculate the "advantage" as: $A(\mathbf{xy}) = [RM(\mathbf{xy}) - \beta \cdot KL(\mathbf{xy})] - V(\mathbf{x})$. This is a measure of whether the policy function should incentivize or avoid this

[2] We often use MSE loss as the loss function

[3] Recalling that a state is defined by the *exact* sequence of tokens prior in the sequence.

type of response. As previously mentioned, in Section 2.1, the reward model, $RM(\mathbf{xy})$, gives a numerical estimate of how much the user likes (or will like) the response [4]. Here is where we subtract a weighted amount of the previously computed KL-divergence (see Section 2.2) to penalize the model for straying too far from the original parameters.

If we find that $RM(\mathbf{xy}) > V(\mathbf{x})$, then this means the generated response was good, and the policy should be adapted to make generating this response more likely. If $RM(\mathbf{xy}) < V(\mathbf{x})$, it means that this response was considered bad, and the policy should avoid such patterns in its responses.

2.4 Policy Ratio

PPO is an off-policy method. This means that we collect the sample prompts from the policy iteration $\pi_{\theta_{j-1}}$, but we reuse this data for π_{θ_j}. In order to handle distribution shift, we add a correction term. Recall that the core premise of updating RL agents is to calculate gradients based on the function

$$\mathbb{E}_{y\sim\pi_\theta(\cdot|\cdot)}\left[RM(\cdot,y)\cdot\nabla_\theta \ln(\pi_\theta(y|\cdot)\right] \tag{7}$$

When multiplying with the off-policy correction ratio, we can recover an equivalent form for the PPO loss (although using Advantage rather than RM directly, etc). Nonetheless, we do want to resample periodically, otherwise the shift can overcome the correction term and lead to degraded model performance.

Normally, importance sampling correction would be written as the first RHS below, but we prefer the second:

$$c = \frac{\pi_{\theta_j}(\mathbf{y}|\mathbf{x})}{\pi_{\theta_{j-1}}(\mathbf{y}|\mathbf{x})} \tag{8}$$

$$= \exp\left[\ln(\pi_{\theta_j}(\mathbf{y}|\mathbf{x})) - \ln(\pi_{\theta_{j-1}}(\mathbf{y}|\mathbf{x}))\right] \tag{9}$$

We prefer to use this format for the calculations as it aligns better with the values we computed in Section 2.2 and allows us to reuse our computations.

2.5 Total Loss and Backpropagation

PPO loss: From this point we just apply the scaling factor of the off-policy correction to the calculated Advantage to create the PPO loss. Note that we have π_{θ_t} appearing at multiple locations. Recall that we can expand the PPO loss to become:

$$\mathcal{L}_{PPO} = clip(c(\mathbf{x},\mathbf{y}))\cdot A(\mathbf{xy}) \tag{10}$$

$$= clip\left(\frac{\pi_{\theta_j}(\mathbf{y}|\mathbf{x})}{\pi_{\theta_{j-1}}(\mathbf{y}|\mathbf{x})}\right)\cdot([RM(\mathbf{xy}) - \beta\cdot KL(\mathbf{xy})] - V(\mathbf{x})) \tag{11}$$

$$= clip\left(\frac{\pi_{\theta_j}(\mathbf{y}|\mathbf{x})}{\pi_{\theta_{j-1}}(\mathbf{y}|\mathbf{x})}\right)\cdot\left(\left[RM(\mathbf{xy}) - \beta\cdot\sum_t\left[\ln(\pi_\theta(\mathbf{y}|\mathbf{xy_{i<t}}) - \ln(\pi_{ref}(\mathbf{y}|\mathbf{xy_{i<t}})\right]\right] - V(\mathbf{x})\right) \tag{12}$$

[4] Creation of RMs is a whole separate topic!

This involves the policy π_θ as part of the log-prob in two places: the off-policy correction and the KL divergence. This directly affects the logits that the model produces.

Value function estimation: We also have to train V using the values we see from RM. This is done usually with MSE loss. This impacts the model approximating V only, and does not impact the policy function (and thus not the produced logits).

Exploration term: Lastly, we add an exploration term into the loss to encourage the model to try new responses. It is formulated as:

$$\mathcal{L}_{expl} = -\sum_t \left[\pi_\theta(\mathbf{y}_t|\mathbf{x}\mathbf{y}_{i<t}) \cdot \ln\left(\pi_\theta(\mathbf{y}_t|\mathbf{x}\mathbf{y}_{i<t})\right)\right] \tag{13}$$

which is the stochastic, discrete approximation of entropy for the system. We want to encourage exploration (i.e: encourage entropy in generated responses), so we want to increase the loss when entropy is low, and decrease it when entropy is high. Thus, we add a negative scaled version of the entropy to the loss. We emphasize this in the formula by writing: $\mathcal{L} = \mathcal{L}_{PPO} + ... + -1 \cdot \alpha_2 \cdot \mathcal{L}_{expl}$. This term also impacts the policy model π_θ during backpropagation, giving us a total of three instances where the policy function appears in our final loss. Notably however, the KL-divergence in the calculation of Advantage does not incur gradient tracking and backpropagation as the Advantage is calculated offline. Some works have added a separate KL-divergence term into the loss which does add to the computation graph.

It can be unclear how the information from the Reward/Advantage is being used to update the model. After all, it does not contribute to backprop and is not part of the computation graph. The key insight is that the off-policy correction ratio, r, is a measure of the change of the parameters of the network, and thus contains all the necessary information to update model weights. The Advantage simply tells us how much we should continue to follow the change proposed by the updated weights.

3 Glossary of Symbols

1. $\mathbf{x}$: the prompt given to a LM
2. $\mathbf{y}$: the LM generated response to a given prompt
3. $\pi_{\mathbf{ref}}$: reference model: a frozen version of the pre-finetuning model
4. π_θ: policy model: the model that is actively being matched to engineer preferences
5. KL : the KL-divergence between π_θ and π_{ref}

6. RM: reward model: the model that outputs a numerical score to represent the engineer's preference for each prompt
7. V: the value function for a given state. Do not confuse this with the Value matrix from Q, K, V in attention calculations.
8. A: advantage : a numerical measure of how good or bad a response is based on the estimates of the RM and V functions
9. c : importance sampling correction term : a correction since we are sampling from the distribution of π_{ref} but the loss is computed using π_θ
10. $\mathcal{L}_{PPO}$: the PPO loss corresponding to how well the model performed
11. $\mathcal{L}_V$: the loss used to train the value function
12. $\mathcal{L}_{expl}$: the loss used to encourage exploration
13. $\mathcal{L}$: the total loss used to train the model