Reinforcement Learning from Human Feedback (RLHF): An Accessible Explanation

Introduction

Large language models (LLMs) such as ChatGPT began as systems trained on huge amounts of text. Their first job is simple: predict the next word in a sequence. From this humble objective, surprising fluency emerges. But fluency is not the same as helpfulness, safety, or honesty. That is where reinforcement learning from human feedback (RLHF) comes in.

This handout explains RLHF in a way that does not assume deep mathematical background. We will repeat core ideas often, use minimal notation, and emphasize intuition. Our goal is to understand why RLHF matters, how it works at a high level, and what makes it different from the supervised learning you may have already seen.

1 Pretraining: Predict the Next Word

How the base model works

A large language model begins by learning to predict the next word from context.

- Input: a sequence of words (the context).
- Output: a set of raw scores (called *logits*) for every word in the vocabulary.
- The logits are passed through the *softmax* function, which converts them into probabilities.
- The model then chooses one of the words, usually by sampling according to those probabilities.

Softmax refresher

If the logits for two words are z_1 and z_2, their probabilities are:

$$p_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2}}, \qquad p_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2}}.$$

Softmax ensures that every probability is between 0 and 1 and that all probabilities sum to 1.

Supervised training with cross-entropy

During pretraining, the model knows the "true" next word from its dataset. Suppose the true next word is "red." We represent this as a *one-hot vector*—all zeros except a 1 at the position for "red."

The loss function is *cross-entropy*: it punishes the model if the probability assigned to the true word is too low.

The parameters of the network, denoted θ, are updated by backpropagation so that next time, the model makes "red" more likely in that context.

This process, repeated billions of times, produces a fluent model.

2 The Problem: Fluency is Not Enough

A pretrained model can generate human-like text, but it does not know how to be *helpful*, *safe*, or *truthful*.

- It may eagerly provide dangerous instructions.
- It may invent confident but wrong answers.
- It may ramble when asked to summarize.

Why? Because the pretraining objective was only: "sound like text from the internet." It was never: "be useful to a human asking a question."

3 Reinforcement Learning from Human Feedback (RLHF)

RLHF introduces a new element: a *reward model*. This model is trained not to predict the next word, but to score entire answers according to human preference.[1]

How to build a reward model

1. Give the base model a prompt.
2. Generate two or more answers (A, B, ...).
3. Ask humans: which answer is better?
4. Train a smaller model so that it assigns a higher score to the preferred answer.

[1] An earlier method – and one that is still sometimes useful – is called supervised fine tuning (SFT). The idea is to collect good examples of answers and train the model (somehow) to imitate them. SFT helps, but it has two serious problems:

1. **Coverage problem:** Humans cannot provide good answers for every possible input.
2. **One-answer problem:** Many prompts have multiple acceptable answers. SFT forces the model to imitate only one.

Over time, this reward model learns to generalize: given a new prompt and answer, it can estimate how much humans would like it.

Why this helps

Humans do not have to label every answer forever. They only provide enough comparisons to train the reward model. The reward model then acts as a stand-in for human judgment.

4 Reinforcement Learning Updates

Now we come to the heart of RLHF: using the reward model's scores to update the large model.

The challenge

In supervised training, every token has a known "true" next word. In RLHF, we do not know the true next token. We only know whether the *whole answer* was good or bad.

How can one scalar reward update many token choices? This is the *credit assignment problem.*

The solution: policy gradients

Think of the model as a policy: it assigns probabilities to actions (tokens). The key update rule is:

$$\text{Loss}(\theta) = -R \cdot \sum_{t=1}^{T} \log p_\theta(a_t \mid s_t),$$

where:

- R is the scalar reward for the whole sequence,
- a_t is the token actually chosen at step t,
- s_t is the context at step t,
- $p_\theta(a_t \mid s_t)$ is the softmax probability the model assigned to a_t.

In words:

> Multiply the reward by the sum of the log-probabilities of the chosen tokens. Take the negative of that. That is the loss to minimize with backpropagation. No more cross-entropy!

Intuition

- If reward is positive: increase the probabilities of the tokens that were chosen.
- If reward is negative: decrease them.

Notice the difference from the supervised learning we did to train the model in the first place:

- In supervised learning, the target is a one-hot vector.
- In RLHF, the "target" is the reward applied to the actual sequence chosen.

5 A Tiny Example

Suppose the vocabulary has only two tokens: A and B.

- Current probabilities: $p(A) = 0.5$, $p(B) = 0.5$.
- The model samples A.
- The reward model assigns reward $R = +2$.

Update step:

$$\Delta z_A = \alpha \cdot R \cdot (1 - p(A)) = \alpha,$$
$$\Delta z_B = \alpha \cdot R \cdot (-p(B)) = -\alpha,$$

where α is the learning rate.

Interpretation:

- The logit for A goes up.
- The logit for B goes down.

Next time, $p(A)$ will be a bit larger than 0.5, and $p(B)$ a bit smaller.

If the reward had been negative, the adjustments would go in the opposite direction.

6 Key Phrase to Remember

> In RLHF, you do not reward the top choice. You reward what the model actually did.

This is the crucial idea that allows exploration and improvement.

7 Why PPO (Briefly)

I really wanted to discuss Proximal Policy Optimization (PPO). But Professor Chandler suggested that this was complicated enough already and I should just relegate it to a footnote. So, with some reluctance, I have done so.[2]

8 What RLHF Achieves

After pretraining, a model is fluent but not aligned.

After SFT, it can imitate a few good examples.

After RLHF, it can generalize: producing outputs that are consistently more helpful, safe, and aligned with human preferences.

This is what turned GPT-style models from autocomplete machines into useful assistants.

Summary

1. Pretraining: predict the next word (cross-entropy with one-hot targets).
2. Supervised fine-tuning: imitate human-written answers (still cross-entropy).
3. Reward model: trained from human comparisons to score answers.
4. RLHF: update the big model by nudging probabilities of chosen tokens up or down based on the sequence reward.

And again, the golden phrase:

> Reinforcement learning rewards what the model actually did and what we ended up liking, not just what would have been the top choice.

[2]Proximal Policy Optimization (PPO) is a reinforcement learning algorithm used to keep the updates stable. Without it, the model might drift too far from its pretrained state or collapse into trivial answers.