7 Reward Modeling

Reward models are core to the modern approach to RLHF. Reward models broadly have been used extensively in reinforcement learning research as a proxy for environment rewards [54]. The practice is closely related to inverse reinforcement learning, where the problem is to approximate an agent's reward function given trajectories of behavior [79], and other areas of deep reinforcement learning. Reward models were proposed, in their modern form, as a tool for studying the value alignment problem [32].

The most common reward model predicts the probability that a piece of text was close to a "preferred" piece of text from the training comparisons. Later in this section we also compare these to Outcome Reward Models (ORMs) that predict the probability and a completion results in a correct answer or a Process Reward Model (PRM) that assigns a score to each step in reasoning. When not indicated, the reward models mentioned are those predicting preference between text.

7.1 Training Reward Models

There are two popular expressions for how to train a standard reward model for RLHF – they are numerically equivalent. The canonical implementation is derived from the Bradley-Terry model of preference [80]. A Bradley-Terry model of preferences measures the probability that the pairwise comparison for two events drawn from the same distribution, say i and j, satisfy the following relation, $i > j$:

$$P(i > j) = \frac{p_i}{p_i + p_j} \tag{9}$$

To train a reward model, we must formulate a loss function that satisfies the above relation. The first structure applied is to convert a language model into a model that outputs a scalar value, often in the form of a single classification probability logit. Thus, we can take the score of this model with two samples, the i and j above are now completions, y_1 and y_2, to one prompt, x and score both of them with respect to the above model, r_θ.

The probability of success for a given reward model in a pairwise comparison, becomes:

$$P(y_1 > y_2) = \frac{\exp(r(y_1))}{\exp(r(y_1)) + \exp(r(y_2))} \tag{10}$$

Then, by maximizing the log-likelihood of the above function (or alternatively minimizing the negative log-likelihood), we can arrive at the loss function to train a reward model:

$$
\begin{aligned}
\theta^* = \arg\max_\theta P(y_w > y_l) &= \arg\max_\theta \frac{\exp(r_\theta(y_w))}{\exp(r_\theta(y_w)) + \exp(r_\theta(y_l))} \\
&= \arg\max_\theta \frac{\exp(r_\theta(y_w))}{\exp(r_\theta(y_w))\left(1 + \frac{\exp(r_\theta(y_l))}{\exp(r_\theta(y_w))}\right)} \\
&= \arg\max_\theta \frac{1}{1 + \frac{\exp(r_\theta(y_l))}{\exp(r_\theta(y_w))}} \\
&= \arg\max_\theta \frac{1}{1 + \exp(-(r_\theta(y_w) - r_\theta(y_l)))} \\
&= \arg\max_\theta \sigma\left(r_\theta(y_w) - r_\theta(y_l)\right) \\
&= \arg\min_\theta -\log\left(\sigma\left(r_\theta(y_w) - r_\theta(y_l)\right)\right)
\end{aligned} \tag{11}
$$

The first form, as in [3] and other works:

$$
\mathcal{L}(\theta) = -\log\left(\sigma\left(r_\theta(x, y_w) - r_\theta(x, y_l)\right)\right) \tag{12}
$$

Second, as in [17] and other works:

$$
\mathcal{L}(\theta) = \log\left(1 + e^{r_\theta(x, y_l) - r_\theta(x, y_w)}\right) \tag{13}
$$

7.2 Architecture

The most common way reward models are implemented is through an abstraction similar to Transformer's `AutoModelForSequenceClassification`, which appends a small linear head to the language model that performs classification between two outcomes – chosen and rejected. At inference time, the model outputs the *probability that the piece of text is chosen* as a single logit from the model.

Other implementation options exist, such as just taking a linear layer directly from the final embeddings, but they are less common in open tooling.

7.3 Implementation Example

Implementing the reward modeling loss is quite simple. More of the implementation challenge is on setting up a separate data loader and inference pipeline. Given the correct dataloader, the loss is implemented as:

```
import torch.nn as nn
rewards_chosen = model(**inputs_chosen)
rewards_rejected = model(**inputs_rejected)

loss = -nn.functional.logsigmoid(rewards_chosen - rewards_rejected).mean
    ()
```

Note, when training reward models, the most common practice is to train for only 1 epoch to avoid overfitting.

7.4 Variants

Reward modeling is a relatively under-explored area of RLHF. The traditional reward modeling loss has been modified in many popular works, but the modifications have not solidified into a single best practice.

7.4.1 Preference Margin Loss

In the case where annotators are providing either scores or rankings on a Likert Scale, the magnitude of the relational quantities can be used in training. The most common practice is to binarize the data direction, implicitly scores of 1 and 0, but the additional information has been used to improve model training. Llama 2 proposes using the margin between two datapoints, $m(r)$, to distinguish the magnitude of preference:

$$\mathcal{L}(\theta) = -\log\left(\sigma\left(r_\theta(x, y_w) - r_\theta(x, y_l) - m(r)\right)\right) \tag{14}$$

Note that in Llama 3 the margin term was removed as the team observed diminishing improvements after scaling.

7.4.2 Balancing Multiple Comparisons Per Prompt

InstructGPT studies the impact of using a variable number of completions per prompt, yet balancing them in the reward model training [3]. To do this, they weight the loss updates per comparison per prompt. At an implementation level, this can be done automatically by including all examples with the same prompt in the same training batch, naturally weighing the different pairs – not doing this caused overfitting to the prompts. The loss function becomes:

$$\mathcal{L}(\theta) = -\frac{1}{\binom{K}{2}} \mathbb{E}_{(x, y_w, y_l)\sim D} \log\left(\sigma\left(r_\theta(x, y_w) - r_\theta(x, y_l)\right)\right) \tag{15}$$

7.4.3 K-wise Loss Function

There are many other formulations that can create suitable models of human preferences for RLHF. One such example, used in the popular, early RLHF'd models Starling 7B and 34B [81], is a K-wise loss function based on the Plackett-Luce model [82].

Zhu et al. 2023 [83] formalizes the setup as follows. With a prompt, or state, s^i, K actions $(a_0^i, a_1^i, \cdots, a_{K-1}^i)$ are sampled from $P(a_0, \cdots, a_{K-1}|s^i)$. Then, labelers are used to rank preferences with $\sigma^i : [K] \mapsto [K]$ is a function representing action rankings, where $\sigma^i(0)$ is the most preferred action. This yields a preference model capturing the following:

$$P(\sigma^i|s^i, a_0^i, a_1^i, \dots, a_{K-1}^i) = \prod_{k=0}^{K-1} \frac{\exp(r_{\theta\star}(s^i, a_{\sigma^i(k)}^i))}{\sum_{j=k}^{K-1} \exp(r_{\theta\star}(s^i, a_{\sigma^i(j)}^i))} \tag{16}$$

When $K = 2$, this reduces to the Bradley-Terry (BT) model for pairwise comparisons. Regardless, once trained, these models are used similarly to other reward models during RLHF training.

7.5 Outcome Reward Models

The majority of *preference tuning* for language models and other AI systems is done with the Bradley Terry models discussed above. For reasoning heavy tasks, one can use an Outcome Reward Model (ORM). The training data for an ORM is constructed in a similar manner to standard preference tuning. Here, we have a problem statement or prompt, x and two completions y_1 and y_2. The inductive bias used here is that one completion should be a correct solution to the problem and one incorrect, resulting in (y_c, y_{ic}).

The shape of the models used is very similar to a standard reward model, with a linear layer appended to a model that can output a single logit (in the case of an RM) – with an ORM, the training objective that follows is slightly different [84]:

> [We] train verifiers with a joint objective where the model learns to label a model completion as correct or incorrect, in addition to the original language modeling objective. Architecturally, this means our verifiers are language models, with a small scalar head that outputs predictions on a per-token basis. We implement this scalar head as a single bias parameter and single gain parameter that operate on the logits outputted by the language model's final unembedding layer.

To translate, this is implemented as a language modeling head that can predict two classes per token (1 for correct, 0 for incorrect), rather than a classification head of a traditional RM that outputs one token for the entire sequence. Formally, following [85] this can be shown as:

$$\mathcal{L}_{\text{CE}} = -\mathbb{E}_{(s,r)\sim\mathcal{D}}[r \log p_\theta(s) + (1-r)\log(1-p_\theta(s))] \tag{17}$$

where $r \in 0, 1$ is a binary label where 1 applies to a correct answer to a given prompt and 0 applies to an incorrect, and $p_\theta(s)$ is the scalar proportional to predicted probability of correctness from the model being trained.

These models have continued in use, but are less supported in open-source RLHF tools. For example, the same type of ORM was used in the seminal work *Let's Verify Step by Step* [44], but without the language modeling prediction piece of the loss. Then, the final loss is a cross entropy loss on every token predicting if the final answer is correct.

Given the lack of support, the term outcome reward model (ORM) has been used in multiple ways. Some literature, e.g. [85], continues to use the original definition from Cobbe et al. 2021. Others do not.

7.6 Process Reward Models

Process Reward Models (PRMs), originally called Process-supervised Reward Models, are reward models trained to output scores at every *step* in a chain of thought reasoning process. These differ from a standard RM that outputs a score only at an EOS token or a ORM that outputs a score at every token. Process Reward Models require supervision at the end of each reasoning step, and then are trained similarly where the tokens in the step are trained to their relevant target – the target is the step in PRMs and the entire response for ORMs.

Here's an example of how this per-step label can be packaged in a trainer, from HuggingFace's TRL [41]:

```
# Get the ID of the separator token and add it to the completions
separator_ids = tokenizer.encode(step_separator, add_special_tokens=False
    )
completions_ids = [completion + separator_ids for completion in
    completions_ids]

# Create the label
labels = [[-100] * (len(completion) - 1) + [label] for completion, label
    in zip(completions_ids, labels)]
```

Traditionally PRMs are trained with a language modeling head that outputs a token only at the end of a reasoning step, e.g. at the token corresponding to a double new line or other special token. These predictions tend to be -1 for incorrect, 0 for neutral, and 1 for correct. These labels do not necessarily tie with whether or not the model is on the right path, but if the step is correct.

7.7 Reward Models vs. Outcome RMs vs. Process RMs vs. Value Functions

The various types of reward models covered indicate the spectrum of ways that "quality" can be measured in RLHF and other post-training methods. Below, a summary of what the models predict and how they are trained.

Table 3: Comparing types of reward models.

Model Class	What They Predict	How They Are Trained	LM structure
Reward Models	Quality of text via probability of chosen response at EOS token	Contrastive loss between pairwise (or N-wise) comparisons between completions	Regression or classification head on top of LM features
Outcome Reward Models	Probability that an answer is correct per-token	Labeled outcome pairs (e.g., success/failure on verifiable domains)	Language modeling head per-token cross-entropy, where every label is the outcome level label
Process Reward Models	A reward or score for intermediate steps at end of reasoning steps	Trained using intermediate feedback or stepwise annotations (trained per token in reasoning step)	Language modeling head only running inference per reasoning step, predicts three classes -1, 0, 1
Value Functions	The expected return given the current state	Trained via regression to each point in sequence	A classification with output per-token

Some notes, given the above table has a lot of edge cases.

- Both in preference tuning and reasoning training, the value functions often have a discount factor of 1, which makes a value function even closer to an outcome reward model, but with a different training loss.
- A process reward model can be supervised by doing rollouts from an intermediate state and collecting outcome data. This blends multiple ideas, but if the *loss* is per reasoning step labels, it is best referred to as a PRM.

7.8 Generative Reward Modeling

With the cost of preference data, a large research area emerged to use existing language models as a judge of human preferences or in other evaluation settings [86]. The core idea is to prompt a language model with instructions on how to judge, a prompt, and two completions (much as would be done with human labelers). An example prompt, from one of the seminal works here for the chat evaluation MT-Bench [86], follows:

```
[System]
Please act as an impartial judge and evaluate the quality of the
     responses provided by two
AI assistants to the user question displayed below. You should choose the
      assistant that
follows the 'users instructions and answers the 'users question better.
     Your evaluation
should consider factors such as the helpfulness, relevance, accuracy,
     depth, creativity,
and level of detail of their responses. Begin your evaluation by
     comparing the two
responses and provide a short explanation. Avoid any position biases and
     ensure that the
order in which the responses were presented does not influence your
     decision. Do not allow
the length of the responses to influence your evaluation. Do not favor
     certain names of
the assistants. Be as objective as possible. After providing your
     explanation, output your
final verdict by strictly following this format: "[[A]]" if assistant A
     is better, "[[B]]"
if assistant B is better, and "[[C]]" for a tie.
[User Question]
{question}
[The Start of Assistant 'As Answer]
{answer_a}
[The End of Assistant 'As Answer]
[The Start of Assistant 'Bs Answer]
{answer_b}
[The End of Assistant 'Bs Answer]
```

Given the efficacy of LLM-as-a-judge for evaluation, spawning many other evaluations such as AlpacaEval [87], Arena-Hard [88], and WildBench [89], many began using LLM-as-a-judge instead of reward models to create and use preference data.

An entire field of study has emerged to study how to use so called "Generative Reward Models" [90] [91] [92] (including models trained *specifically* to be effective judges [93]), but on RM evaluations they tend to be behind existing reward models, showing that reward modeling is an important technique for current RLHF.

A common trick to improve the robustness of LLM-as-a-judge workflows is to use a sampling temperature of 0 to reduce variance of ratings.

7.9 Further Reading

The academic literature for reward modeling established itself in 2024. The bulk of progress in reward modeling early on has been in establishing benchmarks and identifying behavior modes. The first RM benchmark, RewardBench, provided common infrastructure for testing reward models [94]. Since then, RM evaluation has expanded to be similar to the types of evaluations available to general post-trained models, where some evaluations test the accuracy of prediction on domains with known true answers [94] or those more similar to "vibes" performed with LLM-as-a-judge or correlations to other benchmarks [95].

Examples of new benchmarks include multilingual reward bench (M-RewardBench) [96], RAG-RewardBench [97], RMB [98] or RM-Bench [99] for general chat,ReWordBench for typos [100], MJ-Bench [101], Multimodal RewardBench [102], VL RewardBench [103], or VLRMBench [104] for vision language models, Preference Proxy Evaluations [105], and RewardMATH [106]. Process reward models (PRMs) have their own emerging benchmarks, such as PRM Bench [107] and visual benchmarks of VisualProcessBench [108] and ViLBench [109].

To understand progress on *training* reward models, one can reference new reward model training methods, with aspect-conditioned models [110], high quality human datasets [111] [112], scaling [24], extensive experimentation [43], or debiasing data [113].