CS542 (Fall 2025) Programming Assignment 6
Reward Modeling

Due December 1, 2025

Introduction

In this assignment you will be implementing a *reward model* as would be used in RLHF and related LLM preference optimization approaches. Reward modeling frames human feedback as a supervised learning problem: given two candidate model outputs for the same input, the reward model predicts which output humans prefer (or assigns scalar scores). Common training losses include pairwise logistic/cross-entropy on comparisons (Bradley–Terry/ Plackett–Luce-style) and regression on numeric scores. Reward models are a core component of RL from Human Feedback (RLHF).

You will be completing the implementation of a small Bradley–Terry-style reward model using a subset of the CNN/Daily Mail dataset. Each sample consists of a news article, and a correct and incorrect summary of the article. The goal is to train the reward model so that it outputs a higher reward when given the article and the correct summary than when given the article and the incorrect summary. You will need to complete various portions of the reward model implementation, such as the data loading, the response sampling, the forward pass, and the calculation of the pairwise accuracy, and the mean reward. You will train your model for 10 epochs and then provide some observations about its performance and sample test outputs.

You will also do an exercise about the properties of the Bradley–Terry model of pairwise preferences and its generalization, the Plackett–Luce model.

Setup

Please consult the file `PA6_Env.txt` for the setup instructions. You will need to install some dependencies, such as the `SentenceTransformers` package. Therefore, you should create a new environment for the packages required

for this assignment and install the versions indicated in the `PA6_Env.txt` file.

The code has been tested on CUDA, Mac Silicon MPS, and CPU. An efficient solution runs in about 18 minutes on the CPU, and in 1-2 minutes on CUDA and MPS. Note that this is a newly-created assignment and so it's possible that not all the bugs are worked out. Please adhere to the environment description provided to you, and when in doubt, try a department Linux machine.

Assignment: Written Exercise

Before you start writing code, consider the following problem.

In class, we discussed the Bradley-Terry model of pairwise preferences and how the BT assumption (that if $y_A \succ y_B$ and $y_B \succ y_C$, then necessarily $y_A \succ y_C$) leads to underfitting of at least one category of preferences in an *intransitive preference cycle* (where $y_A \succ y_B$ and $y_B \succ y_C$, but $y_C \succ y_A$, as in the game Rock Paper Scissors). We showed this using a generic form of the RLHF/DPO objective:

$$J(\theta) = -\mathbb{E}[\log\sigma\big(RM_\theta(x, y_w) - RM_\theta(x, y_\ell)\big)] \tag{1}$$

Review slides 17–21 in Lecture 26 for the derivation.

We also showed how **ΨPO/IPO** [1], that addresses the strong BT assumption of RLHF/DPO, mitigates this issue for transitive preferences ($y_A \succ y_B$, $y_B \succ y_C$, $y_A \succ y_C$), and non-deterministic preferences ($y_A \succ y_B$, $y_B \succ y_A$), but when it encounters *cyclic preferences*, still can suffer from under-/overfitting to portions of the preference data due the quadratic nature of its objective function:

$$J(\theta) = \mathbb{E}\Big[\Big(h(y, y') - \frac{p^*(y \succ y') - p^*(y' \succ y)}{\beta}\Big)^2\Big] \tag{2}$$

Or, under identity assumptions:

$$J(\theta) = \mathbb{E}\Big[\Big(h(y, y') - \frac{I(y, y')}{\beta}\Big)^2\Big] \tag{3}$$

where $h(y, y')$ is the model's learned human reward difference between responses y and y', respectively. Review slides 23–29 in Lecture 26 for the derivation and discussion.

One potential solution to this is to decouple from the pairwise model of preferences entirely and instead use preference data where for each context

x there is a ranked ordering of responses $\{y_1, y_2, \ldots, y_K\}$. This modelization is called the **Plackett-Luce (PL) model** [2, 3], which generalizes the BT model to ranked orderings of responses, with a preference probability defined by:

$$p^*(\tau|y_1, \ldots, y_K, x) = \prod_{k=1}^{K} \frac{e^{r^*(x, y_{\tau(k)})}}{\sum_{j=k}^{K} e^{r^*(x, y_{\tau(j)})}}, \tag{4}$$

for ranking τ of responses $\{y_1, \ldots, y_K\}$. The reward model RM_θ for r^* can be optimized using an objective function given by:

$$J(\theta) = -\mathbb{E}\Big[\log \prod_{k=1}^{K} \frac{e^{RM_\theta(x, y_{\tau(k)})}}{\sum_{j=k}^{K} e^{RM_\theta(x, y_{\tau(j)})}}\Big] \tag{5}$$

For this portion of the assignment, please discuss *why this modelization can still potentially lead to the same net under-/overfitting effect as the BT model in RLHF/DPO/IPO.*

Formal proofs are acceptable but not required. Equations may be used as necessary. There are multiple possible ways to address why PL models may still suffer from under-/overfitting to parts of the preference data. To receive credit, you need to show at least an intuitive-level knowledge of why this should be the case, expressed in plain language. You do not need to come up with specific text examples of prompts and responses. Everything can be done using variable notation. It may be helpful to additionally specify circumstances under which this effect is *not* present and using a Plackett-Luce modelization leads to optimal fitting of all types of ordering or pairs, in order to expose potential circumstances where under-/overfitting occurs.

In particular, discuss the following:

1. Circumstances (i.e., example orderings that occur in the data) that would lead noisy, non-deterministic, or intransitive preference data;

2. Effects of such circumstances on the optimization objective when they are encountered in data; and

3. Why the optimization (Eq. 5) under such conditions leads to under-/overfitting and unstable training updates (HINT: unstable training updates are typically due to either high-magnitude values of J, or the contribution of specific data to the update going to 0).

Assignment: Coding Portion

Most of the code has been written for you. You are given `reward_model.py` and supporting classes. This reward model is using DistilBERT as the underlying language model. You will train the reward model on a small subset of the CNN/Daily Mail preference dataset. You will need to edit `reward_model.py`, `reward_metrics.py`, and `preference_dataloader.py`. Your job is to complete the implementation of the reward model, including data sampling and calculation of certain metrics.

Specifically, you'll need to:

- Complete the implementation of `sample_preference_pair` (in `preference_dataloader.py`) to create the paired preference data. Each sample in the raw data consists of a news article (the context x) and its summary (the winning response y_w).[1] You need to complete the construction of the paired preference training data by extracting a "bad" summary of the article from the dataset to act as the "losing" training sample y_ℓ. While it would be easy to create a "bad" summary that is just gibberish or random words, the trick here is that the "losing" response has to be a "hard negative", or something that is plausible but wrong. You will do this by computing the cosine similarity of the embedding to the good summary to the embeddings of all other summaries, then selecting the summary embedding that has the highest cosine similarity to the embedding of the good summary (excluding the good summary embedding itself), and storing the text that corresponds to that embedding as the bad summary for the original article x. By doing this you should be able to extract a summary y_ℓ from the dataset that discusses similar entities and events to the good summary y_w, but is not an accurate summary of the article x. You can use `util.cos_sim` for your cosine similarity function, and the starter code already precomputes all embeddings for the train, validation, and test splits of the data, so you do not need to do this yourself (you do need to have `SentenceTransformers` properly installed for this to work, though!).

- Implement `randomize_positions` (in `preference_dataloader.py`). Reward models can suffer from position

[1] For an example sample `ex`, the article is found in `ex["article"]` while the summary is found in `ex["highlights"]`. See `https://huggingface.co/datasets/abisee/cnn_dailymail` for more about how the CNN/Daily Mail dataset is constructed.

bias (i.e., if the winning response always appears first or second in the sample, the model can learn to reward something based on when it sees it, rather than the actual content of the text). To mitigate this, you need to randomly decide if the good summary to an article is the first response ("Response A") or the second ("Response B"). For each data sample, you should choose randomly, with a 50% probability, whether the sample should have the winning response as A or B. You also need to label the sample. The sample should be labeled with a **1** if **Reponse A** is the winning response, or with a **0** if **Response B** is the winner. **Please note and adhere to this convention, because if you don't you're going to have a bad time!!**

- Complete the implementation of `forward_text_pair` (in `reward_model.py`). This is the forward pass that gets the `[CLS]` token embedding of the input as a pooled representation, and then feeds that into the reward modeling head to get the actual reward.

- Complete the implementation of `compute_pairwise_accuracy` (in `reward_metrics.py`). Pairwise accuracy is the percentage of samples where the winning response is correctly predicted. A prediction is correct if the reward for the actual winning response is greater than the reward for the other response (in other words, a correct sample is one where the predicted preference label equals the true preference label). You need to calculate pairwise accuracy over all the samples provided. The function expects as inputs tensors for all predictions and labels over a given data split.

- Complete the implementation of `compute_mean_reward` (in `reward_metrics.py`). This should take in a list of all the rewards computed during a training epoch. You should then compute the mean reward of "winning" and "losing" responses over the train samples. Remember that winning response according to the model is the one with the higher reward, but that *this sample may occur in either the first **or** second position in the pair* and you will need to account for this to get accurate numbers.

With a successful implementation, you will see output like the following print out after each of the 10 training epochs:

```
Epoch 1 | Train loss:  0.6106 | Val pairwise acc:  0.900
Mean train winning response reward:  0.242 | Mean train losing response
```

```
reward:  -0.018
PROMPT: ( cnn ) just when you didn ' t think republicans could top
themselves in finding ways to disrespect the president of the united
states, on monday, 47 republican senators - - that is, all but seven
- - sent a letter to iran that appeared aimed at scuttling president
obama ' s diplomatic efforts to prevent tehran developing nuclear
weapons capacity.  the move comes on
RESPONSE A: timothy stanley :  gop senators ' letter to iranian
leaders seems extraordinary.  but undermining a president ' s foreign
policy is not at all unique, stanley says.  he says both left, right
have gone around administrations to deal with foreign leaders.
RESPONSE B: on monday, 47 republican senators sent an open letter
to iran.  sally kohn :  only most extreme example of dishonor and
disrespect.
Reward A: 0.316 | Reward B: 1.181
Predicted:  B
Correct:  B
```

These outputs provide the raw numbers and texts that you will need to discuss in the write-up accompanying this coding portion of the assignment. A random example from the validation set is chosen before training begins, and its contents and rewards are printed at out each epoch. At the end of training, your final pairwise accuracy on the *test* set is printed out, as well as 5 randomly-selected *correctly labeled* examples from the test set and 5 randomly-selected *incorrectly labeled* test examples. These form the basis of your analysis of model performance.

Submission

Submit *a single .zip file* named `<yourlastname_pa6.zip>`. This file should contain your completed `reward_model.py`, `reward_metrics.py` and `preference_dataloader.py` files, and a PDF containing your answers to the written portion of the assignment and observations of reward model behavior. Separate PDFs for the two written portions are acceptable.

Grading

Grades will be determined as follows:

- **20 points** for your discussion of the Plackett-Luce modelization problem.

- The automated grader is provided to you to run the full battery of sanity checks on your code. These tests are non-exhaustive, so just because your code passes these sanity checks doesn't mean it is bug free or fully correct. Therefore simply passing the autograder is not enough, but doing so gives us some assurance about the correctness of your results! There are 6 sanity checks run though the autograder that apportion points as outlined below:

    1. **3 points** for loading your code error-free. We will be testing your code in an environment like that described in `PA6_Env.txt`. If you wrote your code with the indicated versions of the required dependencies, and if you do not change the template code outside of the prescribed areas, this should be no problem.
    2. **10 points** for a correct `sample_preference_pair` implementation.
    3. **10 points** for a correct `randomize_positions` implementation.
    4. **15 points** for correctly completing the `forward_text_pair` function.
    5. **10 points** for correctly completing `compute_pairwise_accuracy`.
    6. **10 points** for correctly completing `compute_mean_reward`.

- You will also receive **2 points** for the correct submission format. You *must* submit a *single .zip file* as described above.

- The final **20 points** come from the write-up, where you should discuss your observations regarding the behavior of the reward model on the train and validation set during training and on the randomly selected test samples after training. Some things you should discuss:

    – How do the mean winning and losing rewards change during training? How does the change in these values compare to the change in the respective rewards for Reponses A and B of the specific randomly-chosen validation sample?

    – Compare the actual and predicted winning and losing responses from the 10 randomly-chosen test samples that print out after model training. What do you observe about the assigned rewards for the correctly predicted samples vs. the incorrectly predicted samples? Consider both the raw pointwise reward values and the differences between winning and losing rewards.

– Look at the text of the winning and losing responses from the 10 randomly-chosen test samples, particularly the 5 incorrectly labeled samples. What do you observe about the texts of the responses? What, if anything, may have made the incorrectly-labeled samples hard to label?

References

[1] Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. "A general theoretical paradigm to understand learning from human preferences". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2024, pp. 4447–4455.

[2] R Duncan Luce. *Individual choice behavior*. Vol. 4. Wiley New York, 1959.

[3] Robin L Plackett. "The analysis of permutations". In: *Journal of the Royal Statistical Society Series C: Applied Statistics* 24.2 (1975), pp. 193–202.