CS 601.471/671 NLP: Self-supervised Models

Homework 7: Aligning Language Models to Our Intents

For homework deadline and collaboration policy, check the calendar on the course website*

Name: Arijit Nukala and TJ Bai
Collaborators, if any: _____
Sources used for your homework, if any: _____

This assignment will assume that we have strong pre-trained language models. Given this foundation, we will focus on aligning language models to our intents and needs. We will continue to work with GPU processors on our cluster and using them with Slurm queuing system. Amazing!

Homework goals: After completing this homework, you should be comfortable with:

- The theoretical concepts behind aligning LLMs;
- Implementation of the alignment algorithms.

How to hand in your written work: via Gradescope.

Collaboration: You can do this homework as a team of 3 people. Basically, you can write a single solution for both people and upload it as a single PDF and a single `.zip` file as a group.

Typesetting: We strongly recommend typesetting your homework, especially if you have sloppy handwriting! :) We will provide a LaTeX template for homework solutions.

*https://self-supervised.cs.jhu.edu/sp2024/

1 How to Sum like Einstein (Einstein Summation Notation)

Einstein summation is a convention for simplifying expression that includes summation of vectors, matrices or in general tensor. This convention allows a compact statement of lengthy summation operators. At times, using this operator can be even faster than the usual way of implementing summations since PyTorch may execute the summations in a way that is more optimized.

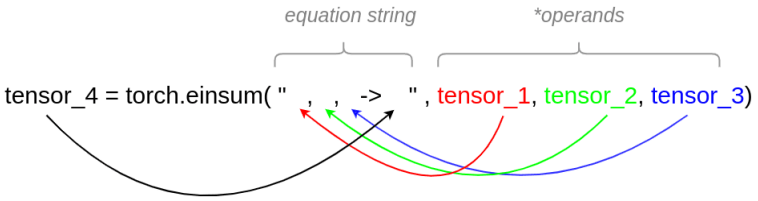
The idea is simple: The sum characters are omitted to improve the overview, and instead, over twice occurring indices are summed. For instance, suppose we want to compute the following:

$$(A.B)_{ij} = \sum_{k=1}^{n} A_{ik}.B_{kj}.$$

This can be written in a more compact form

$$(A.B)_{ij} = \mathcal{A}_{ik}.B_{kj}.$$

With the Einstein notation and the `einsum` function, we can calculate with vectors and matrixces using only a single function: `torch.einsum(equation, *operands)`.



Take this example, matrix multiplication of `A` and `B` can be computed using `einsum` as `torch.einsum("ij,jk->ik", A, B)`. Here, `j` is the *summation index* (summation subscript) and `i` and `k` the *free indices* (output subscripts). Note, the equation may contain white-spaces between the different elements (subscripts, arrow and comma). There are additional nuances in the notation (e.g., ellipsis "...") which can be found in the definition of `einsum(.)`. Since `einsum(.)` is used frequently in PyTorch implementations and its interpretation can be somewhat cryptic, here we want to challenge ourselves with examples of this operator.

	operation	*without einsum*	*with einsum*
1D	$\sum_i x_i y_i$	`x@y`	`np.einsum('i,i', x, y)`
2D	$\sum_i A_{ij} B_{ik}$	`A.T@B`	`np.einsum('ij,ik', A, B)`
3D	$\sum_{i,k} A_{ijk} B_{ik}$	`np.tensordot(A, B, axes=(0, 2),(0, 1))`	`np.einsum('ijk,ik', A, B)`
4D	$\sum_{j,k} A_{ijkl} x_j y_k$	`B = np.tensordot(A, x, axes=(1, 0))` `z = np.tensordot(B, y, axes=(1, 0))`	`np.einsum('ijkl,j,k', A, x, y)`

Figure 1: Caption

For 1D and 2D, the `@` operator wins hands down, but starting from 3D, it becomes apparent that the common PyTorch operators (e.g., dot, matmul and tensordot) are all ill-suited for high dimensions. Here are a few more examples of `einsum`:

Summing along rows/columns:

```
import torch

# Consider a matrix with 2 rows and 3 columns.
u = torch.rand(2,3)

# Use np.einsum(.) operator to sum along the columns.
u_row = torch.einsum('ij->j',u)
```

```
# Use np.einsum(.) operator to sum along the rows.
u_column = torch.einsum('ij->i',u)
```

Here is the resulting variables upon executing this program:

```
>>> print(u)
tensor([[0.4332, 0.0380, 0.2025],
        [0.1818, 0.3093, 0.5738]])


>>> print(u_row)
tensor([0.6150, 0.3473, 0.7763])


>>> print(u_column)
tensor([0.6737, 1.0649])
```

More examples: Below we have more operations that you can do with `einsum`. Feel free to try these yourself to better understand why they're doing:

```
import torch

x = torch.rand(4, 4)
y0 = torch.rand(5)
y1 = torch.rand(4)
z0 = torch.rand(3, 2, 5)
z1 = torch.rand(3, 5, 4)
w = torch.rand(2, 3, 4, 5)
r0 = torch.rand(2, 5)
r1 = torch.rand(3, 5, 4)
r2 = torch.rand(2, 4)
s0 = torch.rand(2, 3, 5, 7)
s1 = torch.rand(11, 3, 17, 5)

# permutations
# this swaps the two dimensions of x
b0 = torch.einsum('ij->ji', x)
# just like the above command, it swaps the ordering of the two dimensions. Note Einsm assumes that
                                                    "a" refers to the dimension before "b".
b1 = torch.einsum('ba', x)
# Here "..." refers to all the dimensions before the last two. The operations swaps the ordering of
                                                    the final two dimensions, without touching the
                                                    earlier dimensions.
b5 = torch.einsum('...ij->...ji', w)
# Here "..." refers to all the dimensions after the first three. The operations swaps the ordering
                                                    of the first three dimensions, without touching
                                                    the earlier dimensions.
b6 = torch.einsum('abc...->cba...', w)

# trace: sums all the values in the matrix
c = torch.einsum('ii', x)
d0 = torch.einsum('ij->', x)
d1 = torch.einsum('xyz->', z0)
d2 = torch.einsum('ijkl->', w)

# Sum the values along a particular matrix
e0 = torch.einsum('ijk->i', z0)
e1 = torch.einsum('ijk->j', z0)
e2 = torch.einsum('ijk->ij', z0)

# Matrix-vector multiplication
f0 = torch.einsum('ij,j->i', r0, y0)
f1 = torch.einsum('i,jki->jk', y1, r1)

# Vector-vector outer product
g0 = torch.einsum('i,j->ij', y0, y1)
g1 = torch.einsum('a,b,c,d->abcd', y0, y1, y0, y1)

# Higher dimensional matrix product
h0 = torch.einsum('bij,bjk->bik', z0, z1)
h1 = torch.einsum('bjk,bij->bik', z1, z0)
```

Now our goal is to implement Self-Attention using this compact notation:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}}\right)\mathbf{V} \in \mathbb{R}^{n\times h}.$$

In the following code, fill in the two missing lines using `einsum` notation.

```
def self_attention(self, x, mask=None):
    assert x.dim() == 3, '3D tensor must be provided'

    # Step 1
    # Extract K, Q, V each with dimensions [batch, tokens, hidden_dim]
    q, k, v = self.to_qvk(x)

    # Step 2
    # QK^T/self.scale_factor
    # Resulting shape: [batch, tokens, tokens]
    # Assume that self.scale_factor is defined outside this function
    # TODO1: Fill in your answer here in in place of the ellipsis
    scaled_dot_prod = torch.einsum('...ij,...kj->...ik', q, k) / self.scale_factor

    if mask is not None:
        assert mask.shape == scaled_dot_prod.shape[1:]
        scaled_dot_prod = scaled_dot_prod.masked_fill(mask, -np.inf)

    attention = torch.softmax(scaled_dot_prod, dim=-1)

    # Step 3
    # The overall attention output
    # TODO2: Fill in your answer here in in place of the ellipsis
    output = torch.einsum('...ij,...jk->...ik', attention, v)

    return output
```

2 Aligning Language Models: Playing with the Formalism

Recently, *reinforcement learning from human feedback* (RLHF) has emerged as a popular technique to optimize and align a language model with human preferences [Ouyang et al., 2022]. RLHF provides a natural solution for optimizing non-differentiable, scalar objectives for language models, and has been the centerpiece of recent state-of-the-art large language models (LLMs). *In this homework, we will review the theory behind RLHF and reduce it into simpler forms.* Let's start with a recap of RLHF:

Reward model. Following the conventional setup [Ziegler et al., 2019], we define a reward r_θ as a scalar function that allows us to train a generative model during RLHF. Rewards are typically calibrated to proxy human preferences/rankings of responses, in the context of an input instruction.

How do you train this reward model? In general, you can use human preferences. Specifically, consider a dataset of human preferences $\mathcal{D}_h = \{(x_i, y_i^+, y_i^-)\}$ is comprised of an input instructions x_i, a pair of responses y_i^+, y_i^- where y_i^+ is preferred over y_i^- by humans. On this human-labeled data, r_θ is trained to assign a higher reward to human-preferred y_i^+ over non-preferred y_i^- in the context of I_i This can be achieved by *maximizing* the following objective:

$$\tilde{J}(\theta; \mathcal{D}_h) = \mathbb{E}_{(x_i, y_i^+, y_i^-) \sim \mathcal{D}_h} \Big[\log \Pr(y_i^+ \succ y_i^- | x_i) \Big], \tag{1}$$

where $\Pr(y \succ y'|x)$ is the preference function (classifier) as a Sigmoid of the difference of rewards:

$$\Pr(y \succ y'|x) := \sigma\Big(r_\theta(y|x) - r_\theta(y'|x)\Big) = \frac{\exp(r_\theta(y|x))}{\exp(r_\theta(y'|x)) + \exp(r_\theta(y|x))},$$

Given the above optimization objective (Eq.1), with the increasing alignment dataset $\mathcal{D}_h$ size, the reward model r_θ becomes a more and more accurate estimate of human preferences.

Reinforcement Learning (policy optimization) with the learned reward. Given a trained reward function r_θ, we use this to train a policy function π that maximizes the reward. For us, this policy function is essentially a language model that given an input text command x generation an output responses y by executing a sequence of samplings. Specifically, during RLHF we optimize π by *maximizing* the following objective:

$$J(\pi) = \mathbb{E}_{\substack{x \sim D \\ y \sim \pi(x)}} \Big[r_\theta(y|x) \Big] - \beta \, D_{\mathrm{KL}}\Big(\pi \| \pi_{\mathrm{ref}}\Big), \tag{2}$$

where D is the collect of input instructions that we would like to handle. $\pi_{\mathrm{ref}}(x)$ is a reference policy (a pre-trained language model) that is never updated. $D_{\mathrm{KL}}(.)$ denotes the average of the KL-divergences between the probabilities produced by π and π_{ref}, for any partial generation.

$$D_{\mathrm{KL}}\big(\pi \| \pi_{\mathrm{ref}}\big) = \mathbb{E}_{x \sim D} \Big[\mathrm{KL}(\pi(.|x) \| \pi_{\mathrm{ref}}(.|x)) \Big]. \tag{3}$$

This is to ensure that the policy being trained π does not diverge from a proper language model (otherwise we have train a policy that maximizes r while generating gibberish). Training a policy function according to $J(\pi)$ objective (Eq.2) gives rise to the PPO algorithm [Schulman et al., 2017, Ouyang et al., 2022], which is the current most-popular algorithm for reinforcement learning with human feedback.

1. Understanding KL-divergence: First, let's focus on the KL-divergence used in Eq.2. The KL-divergence from a distribution $p(x)$ to a distribution $q(x)$ can be thought of as a distance measure. This is just for intuition, though you should check it by understanding its definition. Here is how it is defined for discrete distributions:

$$\mathrm{KL}\left(p \| q\right) = \sum_x p(x) \log \frac{p(x)}{q(x)}. \tag{4}$$

From an information theory perspective, the KL-divergence specifies the number of additional bits required, on average, to transmit values of x if the values are distributed with respect to $p(x)$ but we encode them assuming the distribution $q(x)$.

1.a Show that $\mathrm{KL}(p\|q) \geq 0$ for any choice of p and q.

Answer: *From the Log-sum inequality, we know that* $\sum_x p(x) log \frac{p(x)}{q(x)} \geq (\sum_x p(x)) log \frac{\sum_x p(x)}{\sum_x q(x)}$. *We also know inherently that the sum over each distribution will yield 1. From this inequality, we show:*

$$\begin{aligned} KL(q\|p) &= \sum_x p(x) log \frac{p(x)}{q(x)} \geq (\sum_x p(x)) log \frac{\sum_x p(x)}{\sum_x q(x)} \\ &= \sum_x p(x) log \frac{p(x)}{q(x)} \geq (1) log \frac{1}{1} \\ &= \sum_x p(x) log \frac{p(x)}{q(x)} \geq 0 \blacksquare \end{aligned}$$

We also show that the KL-divergence is always greater than or equal to o another way.

$E\left[\log \frac{p(x)}{q(x)}\right] = E\left[-\log \frac{q(x)}{p(x)}\right] \geq -\log E\left[\frac{q(x)}{p(x)}\right]$ *by Jensen's Inequality.*

$E\left[\frac{q(x)}{p(x)}\right] = \sum_x p(x) \frac{q(x)}{p(x)} = \sum_x q(x) = 1 \implies -\log 1 = 0 \blacksquare$

1.b Show that $\mathrm{KL}(p\|q) = 0$, if and only if $\forall x : p(x) = q(x)$.

Answer:

Forward $\rightarrow$ *If* $KL(p\|q) = 0$, *that means* $\sum_x p(x) log \frac{p(x)}{q(x)} = 0$. *For this to occur,* $log \frac{p(x)}{q(x)} = 0$. *This is only true when* $p(x) = q(x)$ *since* $log(1) = 0$ *and implies* $\forall x : p(x) = q(x)$.

Reverse $\rightarrow$ *If* $\forall x : p(x) = q(x)$, *the KL divergence term simplifies from* $\sum_x p(x) log \frac{p(x)}{q(x)}$ *to* $\sum_x p(x) log \frac{1}{1}$. *This can be further simplified to* $\sum_x p(x) log(1) = \sum_x p(x) \cdot 0$. *Therefore,* $KL(p\|q) = 0$.

1.c Is it true that $\mathrm{KL}(p\|q) = \mathrm{KL}(q\|p)$?

Answer: *This is not true in general. i.e.* $\sum_x p(x) log \frac{p(x)}{q(x)} \neq \sum_x q(x) log \frac{q(x)}{p(x)}$ *because* $p(x) \neq q(x)$ *in most cases.*

1.d Let p and q be multinomial distributions* over indices $I = \{1, \ldots, k\}$. Assume p is defined as $\forall i \in I : p(i) = \theta_i$ and q is defined as:

$$\forall i \in I : q(i) = \begin{cases} \alpha & \text{if } i \text{ is even,} \\ \beta & \text{if } i \text{ is odd.} \end{cases}$$

where $\sum_{i=1}^{k} p(i) = \sum_{i=1}^{k} q(i) = 1$. For simplicity, assume that k is an even integer. Provide a formula for $\mathrm{KL}(q\|p)$.

Answer: $KL(q\|p) = \sum_{i \in I} q(i) \log \frac{q(i)}{p(i)} = \sum_{i \in I,\, even} \alpha \log \frac{\alpha}{\theta_i} + \sum_{i \in I,\, odd} \beta \log \frac{\beta}{\theta_i} = \frac{k}{2}(\alpha + \beta) log \frac{\frac{k}{2}(\alpha\beta)}{\prod_{i \in I} \theta_i}$

2. Understanding RLHF: Next, let's try to unpack PPO to better understand it.

2.a At this point you should have a good understanding of what KL-divergence is. Let's go back to the cryptic term in Eq.3. The KL-divergence here is between the distributions produced by two language models, one that we're training and another one as a reference model. The divergence is computed for each partial generation, i.e., *upon generating each token from* π. That is why we have noted $\pi(.|x)$ instead of $\pi(y|x)$. So far so good? Great!

With this intuition in mind, we can write Eq.3 as following:

$$D_{\mathrm{KL}}(\pi\|\pi_{\mathrm{ref}}) = \mathbb{E}_{x\sim D}\Big[\mathrm{KL}(\pi(.|x)\|\pi_{\mathrm{ref}}(.|x))\Big] = \sum_{y\sim\pi(.|x)}\Big[\log \frac{\pi(y|x)}{\pi_{\mathrm{ref}}(y|x)}\Big]. \tag{5}$$

Note the apparent discrepancy between this simplification and original definition of KL-divergence in Eq.4. Justify this apparent discrepancy.

Answer: *This simplification does not weight each log term with* $\pi(y|x)$ *as is consistent with the definition of KL-divergence. This is a valid choice because weighting would bias the divergence calculation towards the model's own*

*https://en.wikipedia.org/wiki/Multinomial_distribution

distribution when our goal is to capture the discrepancy between generations. This allows for a more direct comparison between the trained and reference model.

2.b Explain why maximizing the objective in Eq.2 will lead to policy functions π that lead to responses that humans tend to rate positively? (Hint: explain the connection to the reward function r_θ and the reference policy π_{ref}).
Answer: *Maximizing the objective corresponds with maximizing the expected reward, where the reward function is learned based off responses that humans tend to rate positively. Additionally, we include a regularization term so that the learned policy does not differ too far from the reference policy, requiring that the learned policy still produces coherent responses and does not overfit.*

2.c A key consideration in aligning chatbots in Eq.2 is the choice of the input prompts $x \sim D$. Basically, all the inputs in this collection get supervised during the alignment process.

Suppose you're building a chat bot that is expected to interact with AT&T customers. You're determined to fix AT&T reputation for having the worst customer service AI. How would you go about building D that would lead to a good outcome? (less than 3 sentences)
Answer: *We would build D to contain a diverse set of possible customer scenarios, especially including particularly aggressive or adversarial customers. As such, the model would be trained to produce responses across all situations that are pleasant. We could source these examples from real chat histories.*

2.d An alternative to PPO (RLHF) algorithm is to do supervised learning (instruction-tuning) on ideal outcome (generations). Explain what are the advantage of RLHF over supervised learning (two advantage each one sentence suffice).
Answer: *RLHF allows the model to generalize to reward signals that correspond more directly with human preferences. Further, RLHF is less reliant on acquiring a large quantity of high quality training data.*

3. DPO algorithm. An alternative approach to the PPO algorithm described above is direct preference optimization or DPO Rafailov et al. [2024] which merges the two steps of RLHF into one step. Assume that we're given a labeled dataset D of preferences, where each input x comes with two output y^+, y^- where y^+ is preferred over y^-. Given a dataset D of preferences, the objective that DPO maximizes as a function of π is as follows:

$$J(\pi) = \mathbb{E}_{(x,y^+,y^-)\sim D}\left[\log\sigma\left(\tau\log\left(\frac{\pi(y^+|x)}{\pi(y^-|x)}\right) - \tau\log\left(\frac{\pi_{\text{ref}}(y^+|x)}{\pi_{\text{ref}}(y^-|x)}\right)\right)\right]. \tag{6}$$

Rafailov et al. [2024] show that upon a perfect optimization of the PPO algorithm the optimal policy coincides with the policy obtained from the optimizer obtained from the objective in Eq.6. Here the implicit reward parameterization is $r(y|x) = \tau\log\frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}$. You can look at Rafailov et al. [2024] for details of their derivations, if you're excited to understand their reduction.

3.a Your friend is claiming that there might be a problem with optimizing this objective in Eq.6. Specifically, your friend claims that, in certain unlucky scenarios the policy model that optimizes $J(\pi)$ might find policy functions that indeed assign low scores to preferred generations, i.e., $\pi(y^+|x) \ll \pi_{\text{ref}}(y^+|x)$, which is not good! What do you think? Explain your rationale (less than 5 sentences).
Answer: *This is certainly a possible failure mode. If the model lowers $\pi(y^+|x)$ but lowers $\pi(y^-|x)$ even more, our objective function still increases. Thus, it is possible to obtain a policy that assigns low probability to preferred generations, just even lower probability to the unpreferred ones.*

3.b Your friend claims that they have identified another failure mode for this optimization. You gasp! Your friend claims that DPO may be prone to over-optimizing preference probability of the preferred over the unpreferred responses: $\pi(y^+|x) \gg \pi(y^-|x)$, thereby effectively ignoring the regularization term based on the reference model. What do you think? Explain your rationale (less than 5 sentences).
Answer: *This is a possible failure mode. $\pi(y^+|x) >> \pi(y^-|x)$ imples that $\frac{\pi(y^+|x)}{\pi(y^-|x)} \to \infty$. $\log\infty \to \infty$. Thus, the regularizer loses effect because it is constant.*

3.c Assuming that you agree with your friend about the above two failure modes of DPO, do you think these failure modes would can also occur in PPO? Justify your answer (less than 5 sentences).
Answer: *While this may be possible with weak regularization in PPO, the KL-regularization term in PPO prevents the learned policy from diverging too much from the reference policy. In effect, this reduces the chance that PPO learns the pathological policies that results in DPO failure modes.*

3.d Your friend modifies the DPO objective in the following form:

$$\hat{J}(\pi) = \mathbb{E}_{(x,y^+,y^-)\sim D}\left[\log\sigma\left(\tau\log\left(\frac{\pi(y^+|x)}{\pi(y^-|x)}\right) - \tau\log\left(\frac{\pi_{\text{ref}}(y^+|x)}{\pi_{\text{ref}}(y^-|x)}\right) - \lambda\max\left(0, \log\frac{\pi_{\text{ref}}(y^+|x)}{\pi(y^+|x)}\right)\right)\right]. \quad (7)$$

Basically, your friend as added an additional loss term inside the Softmax objective that is parameterized with the hyper-parameter λ. Do you think this modification addresses the issues in 3.a and 3.b?
Answer: *This modification is introduced in the paper "Smaug: Fixing Failure Modes of Preference Optimisation with DPO-Positive", which introduces a modified DPO loss that the authors coin "DPOP." This optimization fixes the first issue above because it penalizes the model for decreasing the log-likelihood of the preferred example over the reference generation. However, it doesn't address the second issue because if the preference probability of the preferred example is still very high and higher than the reference, the term* $log\frac{\pi_{ref}(y^+|x)}{\pi(y^+|x)}$ *would be under zero, causing the additional loss term to default to 0. Overfitting may still occur.*

3 Programming

In this programming homework, we will perform RLHF on an encoder-decoder model (T5) using the PPO algorithm. In particular, we will

- Initialize a policy model with supervised fine-tuning on a small set of examples.
- Perform RLHF for the initialized policy model.

Skeleton Code and Structure: The code base for this homework can be found at this GitHub repo under the `hw7` directory. Your task is to fill in the missing parts in the skeleton code, following the requirements, guidance, and tips provided in this pdf and the comments in the corresponding .py files. The code base has the following structure:

- `run_sft.py`: initializes a T5 policy model with supervised fine-tuning.
- `run_rlhf.py`: performs RLHF on the fine-tuned T5 model using PPO.
- `./data`: contains the original ASQA dataset and its human-feedback result.
- `./src`: contains all necessary supporting files for supervised fine-tuning and PPO RLHF.

TODOs — Your tasks include 1) generate plots and/or write short answers based on the results of running the code; 2) fill in the blanks in the skeleton to complete the code. We will explicitly mark these plotting, written answer, and filling-in-the-blank tasks as **TODOs** in the following descriptions, as well as a `# TODO` at the corresponding blank in the code.

Submission: Your submission should contain two parts: 1) plots and/or short answers under the corresponding questions below; and 2) your completion of the skeleton code base, in a `.zip` file. Note: The model checkpoints will be saved under the sub-directory `./checkpoints` and wandb records will be saved under `./wandb`. We have added these two sub-directories in the `.gitignore`, and thus, please be aware not to include them in the submission `.zip` file. n

Overview and Dataset In this programming assignment, you will get your hands dirtier with RLHF training. You will train models using the Huggingface **Transformers** and **Accelerate** libraries, and **PyTorch**. You will be asked to use the famous WandB platform to better monitor and manage your training process. We will use the ASQA dataset [Stelmakh et al., 2022] to do policy model initialization and RLHF training, and its human-feedback dataset [Wu et al., 2023] to do the reward model training. These two datasets are already provided under the `./data` subdirectory.

For the original ASQA dataset, each input ("`text`") is a long question statement consisting of the retrieved Wikipedia passages. The output ("`answer`") is the answer to that open-ended question. Here is an example:


```
{
    "text": "question: Where does university of miami play football games? context: wikipage:
                                                    Hard Rock Stadium text: Hard Rock Stadium\
                                                    n\nHard Rock Stadium ...",
    "answer": [
        "Located in Miami Gardens, Florida, Hard Rock Stadium plays host to the University of
                                                        Miami ..."
    ],
    "passages": [
        {
            "content": "Hard Rock Stadium\n\nHard Rock Stadium is a multipurpose ...",
            "wikipage": "Hard Rock Stadium"
        },
        {
            "content": "For instance, when the Atlanta Braves ...",
            "wikipage": "Hard Rock Stadium"
        }
    ],
    "question": "Where does university of miami play football games?"
},
```

The data comes with training, validation, and testing subsets. We randomly pick 1k instances from the training set to initialize the police model with supervised fine-tuning, and use the full set to do the final RLHF training.

We will use the human-feedback dataset to train a fine-grained reward model that corresponds to **information completeness**. The motivation is that we want LM to provide **complete** answers for a lengthy and complex question. The information completeness score will be computed from the pairwise human preference.

```
"prediction 1": "Bloom is the second studio album ...",
"prediction 2": "Bloom, the second studio album by Australian singer-songwriter ...",
"prediction 3": "Bloom is the second studio album by Australian singer and ...",
"prediction 4": "Bloom is the second studio album by Australian singer and songwriter Troye
                                                    Sivan, ...",
"preference": [0, 0, 1, 0, 1, 1]
```

Specifically, we sample 4 model predictions for each prompt/question, which gives 6 pairs of model predictions. Then pairwise human preferences are collected on a 3-point scale $\{0, 1, 2\}$. For instance, given $\texttt{Pred}_i$ v.s. $\texttt{Pred}_j$, 0 denotes these two predictions have equal overall quality, while 1 denotes $\texttt{Pred}_i$ is better than $\texttt{Pred}_j$ in terms of completeness and 2 means $\texttt{Pred}_j$ is better than $\texttt{Pred}_i$.

Sbatch Scripts We have provided *.sh* scripts of each RLHF step. All scripts are properly configured and should be smoothly running on the Rockfish without any issues of cuda memory or training time.

Python Scripts Arguments In this homework, we have provided all the necessary arguments and set them up to fit jobs on the Rockfish. There will be no **TODOs** asking you to change the arguments. You could check the meaning of each argument from src/transformers/training_args.py.

Suggestions: HW7 programming requires intensive GPU computations; we recommend starting to run on Rockfish as soon as possible to avoid heavy queue traffic before the DDL.

3.1 What is WandB

In this last homework, we will use Weights & Biases (WandB or W&B) which is a platform to help you monitor your training process, visualize live metrics, save prediction results, and experiment collaboratively. We also recommend using it for your final project experiments. To begin with, we first need to sign up here. Once finish sign up, please type

```
> wandb login
```

in the terminal to log in. Following the prompt to copy and paste your API key to the terminal. Then congratulations, you are all set and ready to train your RLHF models now.

3.2 Initialize Policy Model

Inspired by this work [Ramamurthy et al., 2023], we will first use 1K training examples to initialize our policy model, T5-small, with supervised fine-tuning. For this task, we will only use `run_sft.py` file.

1. **TODOs**: Read and complete the missing lines

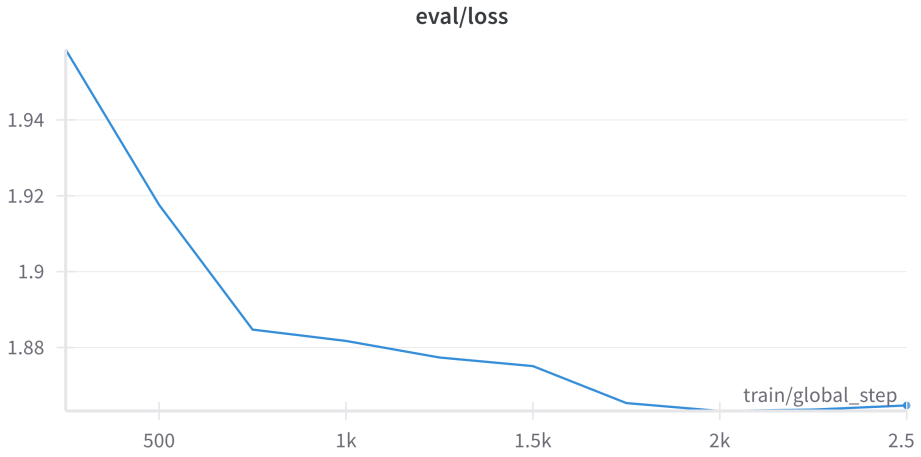
   (a) in the `preprocess_function` function to finish tokenization

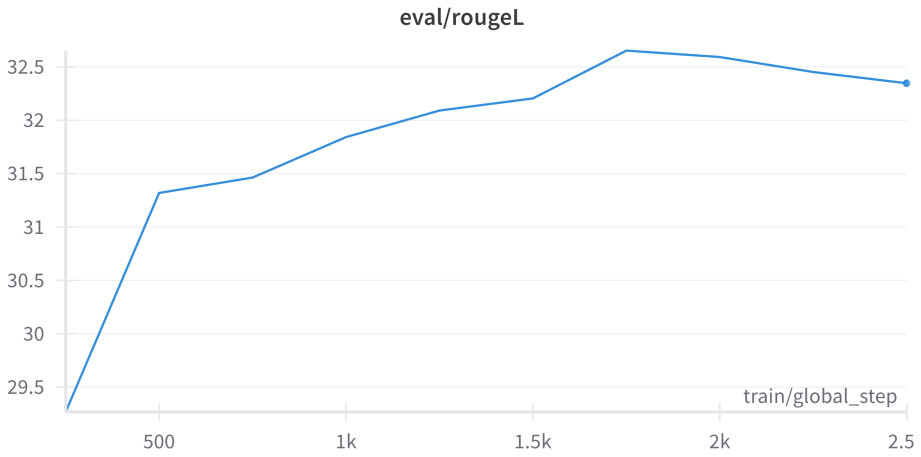
   (b) for the `Seq2SeqTrainer` class to create a trainer instance.

   Hint: we have created and defined all the necessary variables/functions to fill in the class. Your job is to check out the `Seq2SeqTrainer` class and find those pre-defined variables/functions.

2. Now let's initialize our policy model. This step will take approximately 50-60 minutes of running on the Rockfish.

   TODOs: run the `run_sft.sh` script and visualize *eval/rougeL* and *eval/loss* these two metrics over training steps.
   Hint: check out your wandb project homepage.





3. Successfully run the `run_sft.sh` above will automatically run fine-tuned policy model on the testing dataset and save the generated predictions in the file `./checkpoints/sft/generated_predictions.txt`.

   TODOs: Copy and paste the top 3 predictions below and use 1-2 sentences to describe your thoughts on these prediction results and initial model behaviors.
   Hint: Corresponding questions are in the `./data/test.json` file.

   Answer:
   1. Josef Bican has the highest goals in international football. He has the highest goals in international football. He is ranked first in most goals in international football.

   2. Sound of Silence" is a song performed by Australian folk rock duo Simon & Garfunkel. The original artist of Sound of Silence was Dami Im. The original artist was Paul Simon, who was the original artist of the album.

   3. The first apple i phone was made on June 29, 2007, and the first was made on June 29, 2007. The first iPhone was made on September 7, 2005, and the first was made on September 5, 2007. The first iPhone was made on September 7, 2005, and the first was made on September 5, 2007. The first iPhone was made on September 7, 2005, and the first was made on September 5, 2007.

3.3 Reward Model

We provide you with the reward model trained using human preference data introduced above.

TODOs: download and copy the checkpoint directory of the reward model from this link under the `hw7/checkpoints/` directory.

3.4 RLHF

Finally, we will train the policy model using the complete PPO algorithm. In the previous written part, we introduced KL divergence and how it can be used to perform RLHF based on Eq.5. This simplified PPO algorithm is widely used in many LLM pre-training processes, such as LLama2 [Touvron et al., 2023], because of its computational efficiency. It directly optimizes the policy model from an old reference policy model. However, this algorithm can have high variance, leading to unreliable and unexpected model behaviors [Zheng et al., 2023]. In this section, instead of using this simplified version, we will implement the complete and original PPO algorithm as introduced by Schulman et al. [2017]. The original PPO uses an extra value model to estimate the value of the state and optimizes the policy model with a PPO-clipped surrogate training objective Schulman et al. [2017], Liu et al. [2023]. The policy model and the value are trained jointly by using the value model's output as a baseline or a target for the policy's gradient update. This way, the policy model can learn from both its own experience and the value model's feedback, thus reducing the variance of the policy gradient. However, the complete PPO also have some limitations:

- It introduces a trade-off between bias and variance, as the value model may be inaccurate or inconsistent with the policy model's behavior, which can affect the quality of the policy gradient.
- It increases the complexity and computational cost of the algorithm, as they require two models to be updated simultaneously

3.4.1 Experiment Setup

Specifically, our policy model is based on T5-small and is supervised finetuned on 1K training examples. During RL exploration, we use top-k (k = 20) sampling decoding with temperature = 0.7, which is set based on previous RLHF work [Ramamurthy et al., 2023]. The reference policy model used during RL training is the same initialized T5-small but is never updated. We use another trainable T5-small as the value model, updating it simultaneously to get a good estimate of the current state value (cumulative rewards).

In summary, we need the following three kinds of models to perform RLHF: two **intitalized policy models** as trainable policy model and frozen reference policy model, one **finetuned reward model** to provide rewards, and one **trainable value model** to compute state value/advantages for PPO. Below is the pseudo algorithm used for our RLHF training [Wu et al., 2023]:

Algorithm 1 Fine-grained RLHF with PPO

Input initial policy model $P_{\theta_{\text{init}}}$; initial value model $V_{\psi_{\text{init}}}$; K reward models R_{ϕ_k} trained from human feedback; task prompts $\mathcal{D}$; hyperparameters γ, λ, ϵ, β

policy model $P_\theta \leftarrow P_{\theta_{\text{init}}}$, value model $V_\psi \leftarrow V_{\psi_{\text{init}}}$
for step = 1, ..., M **do**
  Sample a batch $\mathcal{D}_b$ from $\mathcal{D}$
  Sample output sequence $y^n \sim P_\theta(\cdot \mid x^n)$ for each prompt $x^n \in \mathcal{D}_b$
  Compute rewards $\{r_t^n\}_{t=1}^{|y^n|}$ for each sampled output y^n by running R_{ϕ_k}
  Compute advantages $\{A_t\}_{t=1}^{|y^n|}$ and value targets $\{V^{\text{targ}}(s_t)\}_{t=1}^{|y^n|}$ for each y^n with V_ψ
  for PPO iteration = 1, ..., μ **do**
    Update the policy model by maximizing the PPO clipped surrogate objective:

$$\theta \leftarrow \arg\max_\theta \frac{1}{|\mathcal{D}_b|} \sum_{n=1}^{|\mathcal{D}_b|} \frac{1}{|y^n|} \sum_{t=1}^{|y^n|} \min\left(\frac{P_\theta(a_t \mid s_t)}{P_{\theta_{\text{old}}}(a_t \mid s_t)} A_t,\ \text{clip}(v_t,\ 1-\varepsilon,\ 1+\varepsilon) A_t \right)$$

    Update the value model by minimizing a square-error objective:

$$\psi \leftarrow \arg\min_\psi \frac{1}{|\mathcal{D}_b|} \sum_{n=1}^{|\mathcal{D}_b|} \frac{1}{|y^n|} \sum_{t=1}^{|y^n|} \left(V_\psi(s_t) - V^{\text{targ}}(s_t) \right)^2$$

  end for
end for

Output P_θ

3.4.2 Experiment

1. **TODOs**: Check out models in the file `run_rlhf.py`. What are the names of those model variables? How many distinct model classes we used and what are their names?

   Answer:
   Variable Names:

   - *ref_policy*
   - *policy*
   - *value*
   - *reward*

   3 Model Classes:

   - *T5 Policy*
   - *T5 Value*
   - *FineGrainedReward*

2. We have arrived at the final step of our RLHF training, yeah!

   TODOs: Read and complete the missing lines after `# TODO: Set up a trainer` in `run_rlhf.py` to create a `PPOTrainer` instance named as `trainer`. We have defined the class `PPOTrainer` in the file `./src/rlhf/ppo.py` file. Please check it out and find all predefined variables/functions that can be used to create its instance. There is no need to create variables/functions on your own.

3. Now let's finally perform RLHF for our policy model. This step will take approximately 50-60 minutes of running on the Rockfish.

   Run the `run_rlhf.sh` script. Once finish training, the prediction results of the testing dataset will be automatically uploaded to your online WandB repo. To access these results, simply visit your WandB homepage and look for the run named `rlhf` under the `hw7` project. Click on it and find the 'file' section in the left panel. Then you can locate the prediction results under

   ```
   > root/media/table/eval_generation
   ```

   Find the file starting with the name "step_100" and
   TODOs: Copy and paste the top 3 predictions to below.and compare the predictions with those in 3.2.3. Do you think that RLHF does help the policy model generate complete and human-preferred answers? If yes, please provide evidence. If no, could you please elaborate on any drawbacks of our current approach/setting and suggest any improvements could be made?

   Answer:
   1. Josef Bican has the highest goals in international football. He is ranked first in most goals in international matches. He is ranked first in most goals in international matches. He is ranked first in most goals in international matches. He is ranked first in most goals in international matches.
   2. "Sound of Silence" is a song performed by Australian folk rock duo Simon & Garfunkel. Written by Anthony Egizii and David Musumeci of DNA Songs, it is best known as Australia's entry at the Eurovision Song Contest 2016 which was held in Stockholm, Sweden, where it finished 2nd, receiving a total of 511 points. The album was initially unsuccessful, so Paul Simon moved to London, England, and Art Garfunkel continued his studies at Columbia University in their native New York City, before reuniting in late 1965. The album's title is a slight modification of the title of the title of the first major hit, "The Sound of Silence" is a song performed by Australian recording artist Dami Im. The album was re-released in January 1966, adding electric guitars, bass guitar and a drum kit), and reached 30 on the Billboard'
   3. The first apple i phone was made on June 29, 2007, and the first iPhone was released later that year. The first iPhone was made on June 29, 2007, and the first iPhone was made on June 11, 2007, and the first iPhone was made on September 7, 2005. The iPhone is manufactured in the Shenzhen factory of the Taiwanese company Hon Hai (also known as Foxconn). The first iPhone was made on September 7, 2005, Apple and Motorola released the ROKR E1, the first mobile phone to use iTunes. The iPhone's main competitors in both consumer and business markets were considered to be the LG Prada, LG Viewty, Samsung Ultra Smart F700, Nokia N95, Nokia E61i, Palm Treo 750, Palm Treo 750, Palm Treo 750, Palm Treo 750, Palm Treo 750, Palm Treo 750, Palm Treo 750, HTC Touch,

RLHF does improve generation quality. For example, in the "Sound of Silence" generation example, generation includes information about the song's writers, the song's rise to popularity, and the derivation of the song's name. RLHF also improves the third generation by including information about the iPhone's competitors and manufacturers. In both cases mentioned, the policy model generates answers that provide additional information that would be useful for humans querying the model.
However, the model still has drawbacks, including repeated generation in sentence one and hallucination in sentence three, where the model mentions three dates for the release of the iPhone. The greatest drawback of the current approach is the inconsistency between the value model and policy model, which results in hallucination specifically. We can implement a post-generation filtering mechanism to identify repeated and hallucinated information, remove it, then provide the final result. We may also want to include multiple reward signals in our reward mechanism such as coherence and factual accuracy to further improve outputs. Lastly, training on more examples would improve our results.

Optional Feedback

Have feedback for this assignment? Found something confusing? We'd love to hear from you!

References

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems* (NeurIPS), 2022. URL https://arxiv.org/abs/2203.02155.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019. URL https://arxiv.org/abs/1909.08593.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. ASQA: Factoid questions meet long-form answers. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8273–8288, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.566. URL https://aclanthology.org/2022.emnlp-main.566.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training. *arXiv preprint arXiv:2306.01693*, 2023. URL https://arxiv.org/abs/2306.01693.

Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing: Benchmarks, baselines, and building blocks for natural language policy optimization, 2023. URL https://arxiv.org/abs/2210.01241.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan

Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Secrets of rlhf in large language models part i: Ppo, 2023. URL https://arxiv.org/abs/2307.04964.

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. Don't throw away your value model! making ppo even better via value-guided monte-carlo tree search decoding, 2023. URL https://arxiv.org/abs/2309.15028.