Exploring Reward Design and Policy Gradient Methods for Abstractive Summarization

Rohit Chawla, Rahul Vuggumudi

Motivation

Fine-tuning is an extremely useful tool to optimize performance in language models for a number of downstream tasks. Summarization, one primary example of these downstream tasks, requires an output that effectively balances informativeness with brevity. Traditional summarization methods are usually trained with cross-entropy loss, which aims to minimize the negative log-likelihood when predicting the next token of a sentence. However, this approach encourages the model to predict the exact token without much regard for overall summarization quality, which is typically evaluated through metrics such as ROUGE, BLEU, or even human preferences.

While it may seem simple to optimize for these desired summary metrics, they are non-differentiable, limiting the ability of standard training to do so. On the other hand, reinforcement learning enables us to fine-tune the models based on task-specific rewards, enabling metric alignment and direct optimization for our desired summary qualities. In this project, we aim to explore how reinforcement learning fine-tuning can improve small summarization models under different reward designs.

Approach

We will be using a small (60M-200M) pretrained encoder-decoder model (ex: t5-small) initially, but we may test different model sizes depending on the performance witnessed and training efficiency with regard to our compute constraints.

We will try to fine-tune our model on a couple of different datasets, which include: **- could be changed -** CNN/Daily Mail and XSum, since the former will provide longer, extractive articles, and the latter offers shorter abstractive targets.

We also plan on testing the usage of QLoRA or LoRA in our reinforcement learning fine-tuning process to boost efficiency if our compute/memory constraints require it.

For our reinforcement learning method, we have looked into using state-of-the-art policy gradient methods such as PPO (Proximal Policy Optimization), REINFORCE, GRPO (Group Relative Policy Optimization), and SCST (Self-critical Sequence Training). We would most likely use an RL library like cleanRL or StableBaselines3. We are using proper RL methods for

fine-tuning over something like DPO because we believe our model can produce stronger summaries with the allowed flexibility to design our reward signals. Additionally, we believe that optimizing upon multiple measurable metrics (ROUGE, BLEU, etc) with RL as opposed to preference data with DPO would result in summaries that fit better to our goals.

For our main testing hypotheses, we want to try out different reward function designs to see which allows the model to generate the best summaries. Additionally, the idea of a "best summary" is very subjective, and our reward function designs will include combinations of different metrics and summary properties (more on this in the "Experiments" section below).

Finally, we have ideas for additional steps of complexity that we could include, given that the rest of our project goes extremely well and ahead of schedule. These include:

1) Dynamic reward modeling (changing the reward model after a certain number of epochs based on model performance)

2) Integration of curriculum learning (automatically ramp up the difficulty of examples based on model performance)

Experiments

As mentioned earlier, we plan on designing different reward functions using combinations of different metrics to prioritize certain summary properties. A few example summary properties we might use are: brevity, informativeness, redundancy, hallucination, and fluency**.** We would maintain an evaluation set for final scores with different summarization metrics. We will also use human judgment on the model summaries. Lastly, we can use metrics focused on the specific summary properties to test them.

We also want to compare with a basic supervised fine-tuning approach to see if there is an improvement to support our hypotheses of RL fine-tuning's benefits.

Feasibility

We plan on utilizing Google Colab PRO's A100 GPUs through the student developer program. For efficiency, we are looking into pairing the open-source Unsloth platform (has an optimized LoRA/QLoRA that cuts VRAM and speeds up fine-tuning) with vLLM, a high-throughput inference engine (PagedAttention, continuous batching) that lets us batch RL rollouts. If a single GPU isn't enough, we'll move to OpenRLHF, a Ray + DeepSpeed ZeRO-3 stack with vLLM-backed sampling, so we can scale PPO/GRPO while keeping our custom reward function. We are also prepared to use different model training tricks to make our process more efficient as well if needed. These include: lower batch sizes for memory limits, gradient accumulation to

simulate a bigger batch, LoRA / QLoRA instead of updating all the weights, and starting from a supervised baseline (run for a number of steps before RL).