Training LLMs on Simple, Safe Text

September 24, 2023

1 Introduction

In this project we will explore how to build a LLM that answers questions in a simple and safe (S^2) way. That is, we want an LLM which answers questions in a way which is understandable to schoolchildren and/or adults who are learning English while also not producing any potentially toxic or offensive content. To train the LLM we will use a combination of imitation learning and preference modeling. The two main algorithms we will explore are Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO). In the remainder of this section we will briefly review these methods. We will not go through the detailed implementations of these algorithms (although this may be added in the future) but will instead focus on the overall challenges and benefits of each approach.

1.1 RLHF

RLHF is the most well-known method used to align LLMs with human interests and is behind the success of chatbots such as ChatGPT and Claude. RLHF is roughly given by the following three step procedure:

1. We start by training the model using supervised fine-tuning (SFT). At this step we simply prompt the model with a question in the form "### Human: {question} ###Assistant:" and ask the model to autoregressively generate new text. This is called supervised fine-tuning because we have a reference human-written answer and want to minimize the KL-divergence between the reference text and the model generated text. Despite its name, SFT is essentially the same as pretraining, except we do not compute the loss on the prompt.

2. Next, we perform reward modeling (RM), which is a preference modeling task. In this step, we take another neural net and present it with a question and two answers, one of which is labelled "chosen" and the other one is labelled "rejected". At this step, the goal is to teach the model to identify which answer is preferred over the other, i.e.

to learn human preferences via sequence classification. Typically the reward model is taken to be the LLM from step (1), but we replace the head of the SFTed model with a layer to compute the reward function $r(Q, A)$ and train the model with a loss function such that the model assigns higher rewards to the chosen answer and lower rewards to the rejected answer.

3. Finally, we use the reward model from step (2) to train the SFTed model from step (1) using reinforcement learning. Specifically, the model is trained using Proximal Policy Optimization (PPO). At this step it helps to add a term to the reward modeling objective proportional to the KL-divergence between the SFT model we are training and the original SFT from step (1). The addition of a KL penalty helps ensure that the model trained with RL does not deviate too strongly from the original model and produce low-quality text that simply "hacks" the reward model.

RLHF is clearly a non-trivial algorithm to implement in practice. The simplest step to implement is the SFT since the training objective is essentially identical to the pretraining objective. In addition, if we have a dataset of high enough quality, we arguably do not need to perform many steps of supervised fine-tuning, e.g. in the LIMA paper they achieved remarkable after only training the model with SFT on 1000 examples!

The second step, reward modeling, is simple to implement in principle, but in practice can be very subtle and difficult to get right. For one, we are trying to teach a model what human preferences are by simply ranking two answers. This may be too rough of a metric to actually determine human preferences, e.g. what answer is preferred over the another may depend on how the question is written and who the intended audience is. In addition, what answer is considered the "best" will be biased by the preferences of whoever is ranking the answer, and their preferences may not be universal. In theory, it may also be better to have multiple reward models to judge different aspects of an answer, as opposed to having one reward model which is supposed to determine the quality of the answer in full.[1] However, despite its simplicity, learning human preferences by ranking answers has proven remarkably successful and is also a natural starting point for solving the alignment problem.

To successfully perform reward modeling, we also need to ensure that our preference dataset is of a high-enough quality such that we can use our trained reward model during the reinforcement learning step. The general wisdom currently is that the reward model needs to be achieving accuracies of around 70% (and ideally higher) before we can use it for reinforcement learning. The fact these accuracies are fairly low is a testament to the fact that reward modeling by itself is a non-trivial task! We also need to ensure the preference modeling dataset is of a wide enough scope such that the reward model knows how to properly judge answers that are generated during reinforcement learning. For example, a

[1] We may want separate reward models to separately judge the simplicity, helpfulness, and harmfulness of the answer.

model trained only to judge answers about math may not accurately judge answers about other topics, such as history or politics.

The final step of RLHF, implementing PPO, is more technically challenging than the previous steps. The PPO algorithm requires that we have a value function, a reward model, the policy function (the LLM currently being trained), and the initial SFTed model (which serves as our reference model). Therefore, during PPO we have to keep four separate models in models, and they are all LLMs! For this reason, the PPO step can be fairly memory intensive. To avoid out-of-memory errors, and also spending too much money on GPUs, we either need to use relatively small LLMs and/or use parameter efficient methods such as QLoRA. In addition, it is well-known that training a model with RL is more difficult than training a model using supervised learning and that RL algorithms tend to be more sensitive to hyperparameters and choices of initialization. For this reason, we RL typically requires more hyperparameter optimization than supervised or unsupervised learning.

1.2 DPO

DPO is a newer algorithm designed to achieve the same objective as RLHF using only supervised learning. DPO is a supervised learning algorithm that optimizes the same objective as RLHF, except without needing to train a separate reward model (hence the subtitle of the paper, "Your Language Model is Secretly a Reward Model") and without needing to perform reinforcement learning. To do this, they use an exact map between reward functions to optimal policies such that the loss function on the reward model becomes a loss function on the policy model. Therefore, step (2) of the RLHF algorithm, performing supervised learning on the reward model, turns into a supervised learning problem on the policy function directly, which in our case is the SFTed LLM.

DPO is therefore a two step procedure:

1. Perform supervised fine-tuning on a pre-trained LLM in the same exact way as step one of RLHF.

2. Take the model from step (1) and train it to learn human preferences directly using the DPO loss function.

DPO is clearly a simpler algorithm to implement than RLHF and since we are using supervised learning, it is also expected to be more stable. Given that it is easier to implement, we will start by using DPO to align our LLMs and use the results from DPO as a benchmark for RLHF. In the original DPO paper it was also shown that DPO achieves comparable or better results than RLHF, so we expect DPO will serve as a strong benchmark for RLHF, or any other algorithm to align LLMs.

Are there any downsides to DPO? One potential downside, given its current incarnation, is that there may be cases where we want a separate reward model. This is brought up in the DPO paper, where they mention that having a separate reward model may be useful to label currently unlabelled prompts, and it is less clear if performing self labelling using the LLM (which is also being trained) is equally effective. In addition, as brought up earlier, it may be useful to have multiple reward models to capture different aspects of an answer, e.g. helpfulness vs simplicity. This theoretically can be implemented in RLHF but it is not clear how to do this using DPO directly. Finally, like most implementations of RLHF, the DPO algorithm assigns a single score to an entire answer and does not yet take into account fine-grained human preferences.

2 Datasets

2.1 Sources

The two sources of our datasets are the ELI5 dataset and Simple Wikipedia. In this section we will go into detail on the form of these datasets and in the next section we will explain how we filtered them.

The ELI5 dataset gets its name from the subreddit Explain Like I'm Five (r/ELI5) on reddit.com. The stated goal of the ELI5 subreddit is to provide a place where people can ask questions on a wide variety of topics and receive a layperson-friendly explanation. Commentators are requested to answer a question without assuming "knowledge beyond a typical secondary education program" and in general to keep answers clear and simple. The ELI5 dataset also contains question/answer pairs from the r/AskHistorians and r/AskScience subreddits. The r/AskHistorians is well-known to be a well-moderated community where answers are particularly in-depth and clear. The quality of posts on r/AskScience is also high and users generally receive clear answers to fairly technical questions. We can generally use the ELI5 dataset to train a question/answer model by using the title and body of the post as the question and highly-voted comments as the answers.

Simple Wikipedia is a version of Wikipedia where users are encouraged to write articles using basic English. On the Simple Wikipedia homepage they write: "The Simple English Wikipedia is for everyone, such as children and adults who are learning English." They also encourage users to write shorter sentences, while not necessarily requiring that the articles also be short. In particular, the focus is that the articles are written with simple words and grammar, but do not necessarily contain "basic information". Unlike the ELI5 dataset, we can not use the Simple Wikipedia dataset out of the box to train a question-answering model since articles are not typically written in that style. Instead, we will use GPT-3.5 to generate questions whose answer is contained in the first few paragraphs of the Simple Wikipedia article. This is a very general and powerful idea which allows us to use large LLMs to generate whole new datasets from existing text. A nice feature of this method is that only the question is AI-generated, the answer itself is simply human-written text which has been repurposed for a new task.

2.2 Dataset Types

The goal of

2.3 Filtering

In order to train a model which answers questions in a safe and simple way, we need to filter our

2.4 EDA

3 Model and Training

4 SFT Inference Results

In this section we will summarize the results of Llama-2 models fine-tuned using supervised fine-tuning.

4.1 HuggingFace Leaderboard

First I'll summarize the results of our fine-tuned models versus the Meta-Llama models on the Huggingface Open LLM leaderboard. The leaderboard is a wrapper for the "Eleuther AI Language Model Evaluation Harness". Specifically, the leaderboard measures the 25-shot performance of LLMs on the arc-challenge dataset, the 10-shot performance on the HellaSwag dataset, the 0-shot performance on TurthfulQA, and the 5-shot performance on MMLU. For arc-challenge and HellaSwag performance is measured using the $acc-norm$ metric of EleutherAI, for TruthfulQA they use the mc2 metric, and for MMLU they average the accuracy of the model across tasks in the MMLU dataset. The nice thing about the leaderboard is one can submit either the full model or submit just the adapter layers and give the base model separately. The problem is, sometimes models disappear from the leaderboard and I'm not sure why. Here Llama-2-7b is the base 7 billion model and Llama-2-7b-chat is the Llama-2 model which has undergone RLHF. The remaining models are fine-tuned versions of the base-model. Specifically, they are trained using SFT and QLORA on the ELI5 SFT dataset, the Simple Wikipedia Instruct dataset, or their combination. The second half of the table is the same, except for the 13B parameter model.

When defining the model there are subtleties about how to merge the LoRA adapter weights with the rest of the model. The subtlety arises because in QLoRA we quantize the base model to 4-bit, but need to dequantize these weights to bfloat16 when performing back-propagation for the LoRA adapter layers. When we merge the adapter layers with the base model we have two natural options: either quantize the model to bfloat16 and then merge or quantize the model to 4-bit, dequantize to bfloat16, and then merge. The second option is arguably more natural since during training we are quantizing and then

Model	Average	Arc	HellaSwag	MMLU	TruthfulQA
Llama-2-7b	53.40	53.07	77.74	43.80	38.98
Llama-2-7b-chat	**56.34**	52.90	78.55	**48.32**	**45.57**
Llama-2-7b-ELI5	53.92	53.41	77.90	43.56	40.81
Llama-2-7b-wiki	53.72	**54.35**	78.06	45.35	37.11
Llama-2-7b-ELI5-wiki	55.46	53.75	**78.76**	46.02	43.31
Llama-2-13b	56.90	58.11	80.97	54.34	34.17
Llama-2-13b-chat	59.93	59.04	81.94	54.64	**44.12**
Llama-2-13b-ELI5	**60.61**	**60.41**	**82.58**	**55.86**	43.61
Llama-2-13b-wiki	58.12	59.04	82.33	55.36	35.75
Llama-2-13b-ELI5-wiki	59.43	59.98	82.43	55.41	39.90

Table 1: Results for Llama-2 models on Huggingface Open LLM Leaderboard

.

dequantizing, so we want the model at inference to be as close as possible to the model during training. However, when we quantize and then dequantize we risk losing precision and degrading the model in the process. It is not clear which choice is better and this likely depends on how strong of an effect the LoRA layers have and how sensitive they are to the exact form of weights. On automated benchmarks we have not seen one method give reliably better results than the other.

All that said, to get the above results we quantized and dequantized the 7B model before merging, while for the 13B model we quantized the model to bfloat16 and then merged. For the 7B model we could perform the quantization and dequantization on a 40GB A100. For the 13B model we directly submitted the adapter layers to the HuggingFace leaderboard and used the ungated Llama-2-7b-hf model from NousResearch (the Meta-Llama model is gated and although it can be downloaded, we were not able to use it as a base model on the leaderboard).

In Table 1 we see that of the 7B models, the Meta-Llama-2-7b-chat performs the best on average with the Llama-2 model trained on ELI5 and Simple Wikipedia performing the second best. One surprising thing is the chat model actually performs worse than the base model on the Arc-Challenge dataset, and here the model trained on just Simple Wikipedia QA pairs performs the best. On the HellaSwag dataset the Llama-2-7B model trained on ELI5 and Simple Wikipedia marginally outperforms the chat model, but the difference is likely too small to be statistically significant. Finally, on MMLU and TruthfulQA the 7B chat model performs significantly better than the other models.

Once we go up to 13B parameters we see that the Llama-2 model trained on the ELI5 SFT dataset performs the best on average with the 13B chat model and 13B model fine-tuned on ELI5 + Simple Wikipedia close behind. It is surprising that the model trained on just ELI5 model performs the best, and this result is driven primarily by its improved performance on the TruthfulQA dataset. On the other datasets it barely improves over the model trained on the combined ELI5 and Simple Wikipedia dataset. We are not sure what the cause of this effect is, somehow training the model more on Reddit data makes the model more honest! This result could also be an artifact of a poor choice of hyperparameters, and perhaps with a different learning rate and/or after averaging over initializations the difference would go away or the model trained on the combined dataset would perform better.

4.2 MT(S)-Bench

In this section we will investigate how well our models perform on a new MT-bench like dataset. The idea of MT-bench is to use a LLM, like GPT-4, to judge the output of other, smaller LLMs. There are two types of completion tasks, single-turn and multi-turn. For a single-turn problem, the model is given a prompt and asked to complete it. For a multi-turn problem, the model is prompted with one complete turn of a conversation (i.e. question and answer) and then asked to reply to a new, follow-up question. Given that our models are only trained for single-turn problems, we will see that they do not perform well on multi-turn problems.

Model	Single-Turn score	Multi-Turn score
Llama-2-7b-chat-hf	9.40	8.80
Llama-2-7b-ELI5-wiki-simple-merge	6.683333	3.533333
Llama-2-7b-ELI5-wiki	5.975000	2.966667
Llama-2-7b-wiki	5.108333	1.616667
Llama-2-7b-ELI5	4.116667	1.733333
Llama-2-7b-wiki-simple-merge	2.300000	1.033333
Llama-2-7b-ELI5-simple-merge	2.000000	1.200000
Llama-2-7b-hf	1.066667	1.000000

Next let's look at the readability scores of each model on the single prompt:

Model	FRE	FKG
Llama-2-7b-chat-hf	42.21	12.50
Llama-2-7b-ELI5-wiki-simple-merge	60.55	9.60
Llama-2-7b-ELI5-wiki	53.41	10.20
Llama-2-7b-wiki	91.41	3.90
Llama-2-7b-ELI5	63.19	8.50
Llama-2-7b-wiki-simple-merge	96.48	2.00
Llama-2-7b-ELI5-simple-merge	78.65	4.70
Llama-2-7b-hf	29.86	13.10

4.3 ROUGE and BERTScore

In this section we will investigate how supervised fine-tuning effects the models ROUGE and BERTScores. ROUGE is a well-known automated benchmark that measures n-gam overlap between generated text and the reference text. Since it just looks at n-gram overlaps, and does not take into account semantic content, ROUGE is effectively measuring to what extent the trained model is adopting the vocabulary of the reference text. One advantage of BERTScore is it uses pre-trained encoder models, such as BERT or RoBERTa, to encode the generated and reference text in some high-dimensional vector space and then measures the cosine-similarity between the two vectors. Of course, this also means that BERTScore is more computationally intensive to compute.

In this section we will only perform inference on a small subset (100 QA pairs) of the validation set, for each validation set we randomly sample 100 question and generate the answers.

In the tables below we present the ROUGE and BERTScore metrics for the Llama-2 base model, as well as our fine-tuned models, on the (small) validation sets for our three datasets: ELI5, Simple Wikipedia, and combined dataset. We include the original Llama-2 model as a baseline to measure how much fine-tuning changes the Rouge and BERTScores. In addition, we also include "off-diagonal" elements, where we train a model on one dataset and measure its ROUGE and BERTScores on the validation split of a different dataset. For example, we include cases where we train the model on the ELI5 SFT dataset and then evaluate it on the validation split of the Simple Wikipedia QA dataset. We included these results to serve as additional baselines to see to what extend fine-tuning on *any* QA dataset changes the evaluation metrics. In all cases, the model produces at most 256 new tokens.

It's difficult to interpret or make sense of these results, somehow the base Llama-2 model often outperforms the fine-tuned models and the model trained on just the ELI5 dataset often performs very well on the Simple Wikipedia validation set! It is probably best to take these numbers with a grain of salt. The BERTScore metrics tend to differ by very small amounts, e.g. the difference in F1 scores between the base model and the model trained on ELI5 for the Simple Wikipedia validation set differ by just 0.001. In addition, we of course do not know on what data the original Llama-2 model was trained and if there is data leakage.

Finally, let's look at the Flesch readability ease (FRE) and Flesch-Kincaid grade (FKG) metrics. We see that for the most part, the fine-tuned models have a higher readability score and a lower grade level. There are two noticeable exceptions, the model trained on Simple Wikipedia QA pairs and evaluated on the ELI5 validation set has a higher grade level than the original base model. In addition, the model trained on ELI5 and Simple Wikipedia has a slightly higher grade level than the base model when evaluated on the Simple Wikipedia validation set. In the former case, the grade level went up by 0.451 while the readability also went up by 0.472. Given that a text is considered simpler if its

Dataset	Model	rouge1	rouge2	rougeL	rougeLsum
ELI5	Llama-2-7b	**0.3796**	**0.2432**	**0.3000**	**0.3222**
	Llama-2-7b-ELI5	0.3701	0.2140	0.2736	0.2821
	Llama-2-7b-wiki	0.3575	0.2083	0.2660	0.2762
	Llama-2-7b-ELI5-wiki	0.3702	0.2126	0.2733	0.2819
wiki	Llama-2-7b	0.1923	0.0103	0.0937	**0.1392**
	Llama-2-7b-ELI5	**0.2203**	**0.0125**	**0.0964**	0.1271
	Llama-2-7b-wiki	0.1826	0.0073	0.0879	0.1165
	Llama-2-7b-ELI5-wiki	0.1811	0.0076	0.0885	0.1153
full	Llama-2-7b	0.1944	0.0087	0.0905	**0.1400**
	Llama-2-7b-ELI5	**0.2243**	**0.0118**	**0.0971**	0.1312
	Llama-2-7b-wiki	0.1905	0.0079	0.0889	0.1198
	Llama-2-7b-ELI5-wiki	0.1907	0.0080	0.0894	0.1193

Table 2: Rouge Scores

Dataset	Model	Precision	Recall	F1
ELI5	Llama-2-7b	**0.8429**	0.8754	0.8583
	Llama-2-7b-ELI5	0.8372	**0.8799**	0.8575
	Llama-2-7b-wiki	0.8415	0.8796	**0.8598**
	Llama-2-7b-ELI5-wiki	0.8399	0.8798	0.8590
wiki	Llama-2-7b	**0.7879**	0.8067	0.7970
	Llama-2-7b-ELI5	0.7859	**0.8093**	**0.7971**
	Llama-2-7b-wiki	0.7782	0.8019	0.7898
	Llama-2-7b-ELI5-wiki	0.7783	0.8014	0.7896
full	Llama-2-7b	**0.7940**	0.8092	0.8014
	Llama-2-7b-ELI5	0.7933	**0.8116**	**0.8022**
	Llama-2-7b-wiki	0.7861	0.8048	0.7952
	Llama-2-7b-ELI5-wiki	0.7841	0.8046	0.7941

Table 3: Precision, Recall, and F1 BERTScores

Dataset	**Model**	**FRE**	**FKG**
ELI5	Llama-2-7b	58.2425	10.075
	Llama-2-7b-ELI5	67.6357	8.148
	Llama-2-7b-wiki	58.7145	10.526
	Llama-2-7b-ELI5-wiki	65.6862	8.704
wiki	Llama-2-7b-hf	64.1040	8.172
	Llama-2-7b-ELI5	72.5682	6.988
	Llama-2-7b-wiki	66.5824	8.055
	Llama-2-7b-ELI5-wiki	65.7558	8.239
full	Llama-2-7b	65.0452	8.472
	Llama-2-7b-ELI5	72.5278	7.092
	Llama-2-7b-wiki	66.8147	8.252
	Llama-2-7b-ELI5-wiki	65.8295	8.417

Table 4: Flesch Readability Metrics

grade level is lower and its readability is higher, in this case our automatic metrics are inconclusive. In the latter case, the readability also went up by about 1.6518 points while the grade level went up by 0.067. Once again, in this case our metrics for readability are moving in opposite directions. Finally, from the table we also see that the model trained on just ELI5 tends to produce the simplest text.