

Understanding ChatGPT

From Theory to Implementation

Presenter: Yaowei Zheng

Beihang University

May 5, 2023

Overview

- 1 Introduction
- 2 Preliminaries
- 3 Implementation
- 4 Experiment

Table of Contents

1 Introduction

2 Preliminaries

3 Implementation

4 Experiment

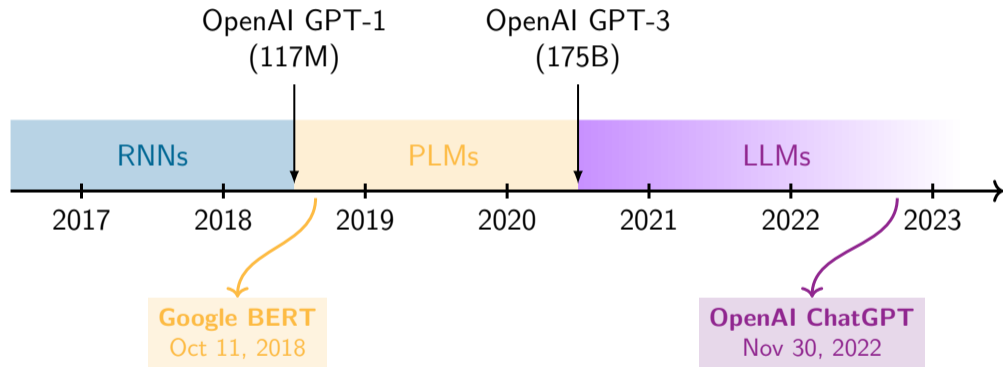
Introduction of ChatGPT

ChatGPT is a sibling model to InstructGPT [1], which is trained to **follow an instruction** in a prompt and provide a detailed response.

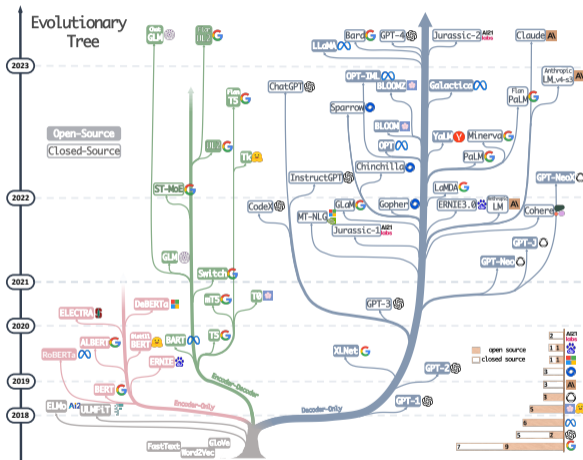
Help me write a short note to introduce myself to my neighbor.

*Hi there,
My name is [Your Name] and I just moved in next door. I wanted to reach out and introduce myself. I'm looking forward to getting to know you and the neighborhood. If you ever need anything or just want to chat, please don't hesitate to stop by or give me a call.
Best,
[Your Name]*

History of Language Models



LLM Families



Illustrated by Yang *et al.*, Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond.

Table of Contents

1 Introduction

2 Preliminaries

3 Implementation

4 Experiment

Transformer

Transformer [2] is based on **attention mechanism**, composed mainly by:

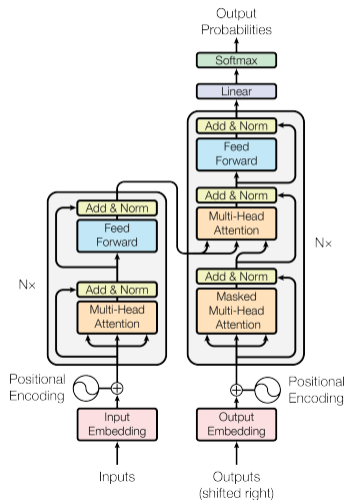
- Multi-head scaled dot-product attention.
- Position-wise feed-forward networks.

It can serve as:

- An encoder model (BERT [3], RoBERTa [4]).
- A decoder model (GPT [5], XLNet [6]).
- An encoder-decoder model (BART [7], T5 [8]).

It outperforms RNN in terms of:

- Computational complexity and **parallelizability**.
- Modeling **long-term dependency** of sequences.



GPT-1: *“Improving Language Understanding by Generative Pre-Training”*

- 117M parameters, 5GB corpus, 12 Layers, single task, released in 2018.
- Applies **auto-regressive models** for unsupervised pre-training.
- Adopts **task-aware input transformations** in supervised fine-tuning.

GPT-2: *“Language Models are Unsupervised Multitask Learners”*

- 1.5B parameters, 40GB corpus, 48 Layers, multitask, released in 2019.
- Achieves zero-shot learning via unsupervised training with **task information**.

GPT-3: *“Language Models are Few-Shot Learners”*

- 175B parameters, 45TB corpus, 96 Layers, multitask, released in 2020.
- Improves zero-shot performance by **scaling up** language models to $100\times$ size.

Instruct GPT

Instruct GPT is trained with 3 different techniques:

Step 1: Make model to follow instructions using supervised fine-tuning (SFT).

- Fine-tune GPT-3 using supervised learning with prompts and demonstrations in a **sequence-to-sequence** manner.

Step 2: Train a reward model with comparison data.

- Rank the outputs from the fine-tuned GPT models and train a reward model (RM) that assigns a **scalar score** to a given input-output pair.

Step 3: Align the model with human preference using reinforcement learning.

- Use this RM as a reward function to fine-tune GPT-3 using **PPO algorithm**.

Supervised Fine-Tuning

They start from an **auto-regressive language model** (like GPT-3) which is trained to predict a probability distribution of the i -th token given the $i - 1$ tokens:

$$P(w_i | w_1, w_2, \dots, w_{i-1}; \theta) \quad (1)$$

Supervised fine-tuning (SFT) is a standard **causal language modeling** task, which minimizes the following negative log-likelihood loss:

$$\mathcal{L}_{\text{SFT}} = - \sum_{i=1}^n \log P(y_i | x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_{i-1}; \theta) \quad (2)$$

where (x_1, x_2, \dots, x_m) is the prompt sequence containing m words and (y_1, y_2, \dots, y_n) is the completion sequence containing n words.

Training Reward Model

Similar to [9], they train the reward model on a dataset of comparisons between two model outputs on the same input, using the **pairwise ranking loss**:

$$\mathcal{L}_{\text{RM}} = - \sum_{(x, y_c, y_r)} \log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r))) \quad (3)$$

where $r_\theta(x, y)$ is the scalar output of the reward model for prompt x and completion y . Besides, y_c is the **preferred completion** out of the pair (y_c, y_r) .

Reinforcement Learning from Human Feedback

Following [9], they fine-tune the SFT model using the **PPO algorithm** [10].

Consider a **bandit problem**, the model is expected to generate a response to a random prompt. Then the reward model produces a reward given the prompt and response.

PPO maximizes the reward with a **per-token KL penalty** between the PPO model and the SFT model, the objective function is:

$$\mathcal{J}(\phi) = \mathbb{E}_{(x,y) \sim \pi_{\phi}^{\text{PPO}}} \left[r_{\theta}(x, y) - \beta \log \frac{\pi_{\phi}^{\text{PPO}}(y|x)}{\pi^{\text{SFT}}(y|x)} \right] \quad (4)$$

where π_{ϕ}^{PPO} is the PPO model, π^{SFT} is the SFT model.

Table of Contents

1 Introduction

2 Preliminaries

3 Implementation

4 Experiment

Required Libraries

Our implementation mainly depends on the following Python libraries:

 PyTorch



Transformers



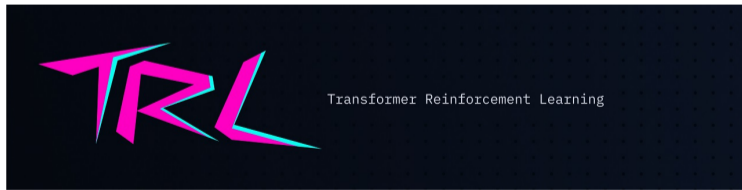
Datasets



Accelerate



PEFT



Loading Pre-trained Model with LoRA Adapters

Firstly, we load the base model using the Transformers library. We also use the PEFT library to enable **parameter-efficient fine-tuning** on **consumer GPUs**.

```
1 from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
2 from peft import get_peft_model, LoraConfig, TaskType
3
4 name = "THUDM/chatglm-6b"
5 peft_config = LoraConfig(
6     task_type=TaskType.SEQ_2_SEQ_LM, inference_mode=False,
7     r=8, lora_alpha=32, lora_dropout=0.1
8 )
9
10 tokenizer = AutoTokenizer.from_pretrained(name, trust_remote_code=True)
11 model = AutoModelForSeq2SeqLM.from_pretrained(name, trust_remote_code=True)
12 model = get_peft_model(model, peft_config)
```

Preparing Instruction-Following Dataset

Before supervised fine-tuning, We load the Stanford Alpaca dataset [11] for **instruction-tuning** using the Datasets library.

```
1 from datasets import load_dataset
2
3 dataset = load_dataset("tatsu-lab/alpaca", split="train")
4 def preprocess_function(ex):
5     model_inputs = {"input_ids": [], "labels": []}
6     for i in range(len(ex)):
7         sid = tokenizer.encode(text=ex[i]["instruction"]+ex[i]["input"])
8         tid = tokenizer.encode(text=ex[i]["output"])
9         input_ids = tokenizer.build_inputs_with_special_tokens(sid, tid)
10        context_length = input_ids.index(tokenizer.bos_token_id)
11        labels = [-100] * context_length + input_ids[context_length:]
12        model_inputs["input_ids"].append(input_ids)
13        model_inputs["labels"].append(labels)
14    return model_inputs
15 dataset = dataset.map(preprocess_function, batched=True)
```

Supervised Fine-Tuning

We use the **Trainer API** provided by Transformers to fine-tune the base model using the instruction-following dataset.

```
1 from transformers import TrainingArguments, DataCollator, Trainer
2
3 training_args = TrainingArguments(output_dir="path_to_checkpoint")
4 data_collator = DataCollator(tokenizer=tokenizer)
5
6 trainer = Trainer(
7     model=model,
8     args=training_args,
9     train_dataset=dataset,
10    tokenizer=tokenizer,
11    data_collator=data_collator
12 )
13 trainer.train()
```

Preparing Comparison Dataset

To train our **reward model**, we load the TL;DR summarization dataset [9] using the Datasets library.

```
1 from datasets import load_dataset
2
3 dataset = load_dataset("CarperAI/openai_summarize_comparisons", split="train")
4 def preprocess_function(ex):
5     model_inputs = {"chosen_ids": [], "reject_ids": []}
6     for i in range(len(ex)):
7         sid = tokenizer.encode(text=ex[i]["prompt"])
8         cid = tokenizer.encode(text=ex[i]["chosen"])
9         rid = tokenizer.encode(text=ex[i]["reject"])
10        cid = tokenizer.build_inputs_with_special_tokens(sid, cid)
11        rid = tokenizer.build_inputs_with_special_tokens(sid, rid)
12        model_inputs["chosen_ids"].append(cid)
13        model_inputs["reject_ids"].append(rid)
14    return model_inputs
15 dataset = dataset.map(preprocess_function, batched=True)
```

Training Reward Model

We inherit the Trainer class to compute the **pairwise ranking loss** [9] for reward model training. We use the predicted score of the last token as the reward.

```
1 from transformers import TrainingArguments, DataCollator, Trainer
2 from trl import AutoModelForCausalLMWithValueHead
3
4 class RMTrainer(Trainer):
5     def compute_loss(self, model, inputs, return_outputs=False):
6         _, _, r_chosen = model(input_ids=inputs["chosen_ids"])
7         _, _, r_reject = model(input_ids=inputs["reject_ids"])
8         loss = -torch.log(torch.sigmoid(r_chosen[-1] - r_reject[-1])).mean()
9         return loss
10 model = AutoModelForCausalLMWithValueHead.from_pretrained(model)
11 training_args = TrainingArguments(output_dir="path_to_rm_checkpoint")
12 data_collator = DataCollator(tokenizer=tokenizer)
13 trainer = RMTrainer(model=model, args=training_args, train_dataset=dataset,
14                     tokenizer=tokenizer, data_collator=data_collator)
15 trainer.train()
```

Reinforcement Learning from Human Feedback

We load the fine-tuned model and reward model using the TRL library as the **actor** and **critic** respectively.

```
1 from torch.optim import AdamW
2 from transformers import DataCollator
3 from trl import PPOConfig, PPOTrainer
4
5 ppo_config = PPOConfig(model_name=model_args.model_name_or_path)
6 model = AutoModelForCausalLMWithValueHead.from_pretrained("ft_checkpoint")
7 rm_model = AutoModelForCausalLMWithValueHead.from_pretrained("rm_checkpoint")
8 data_collator = DataCollator(tokenizer=tokenizer)
9 _params = filter(lambda p: p.requires_grad, model.parameters())
10 optimizer = AdamW(_params, lr=ppo_config.learning_rate)
11
12 ppo_trainer = PPOTrainer(config=ppo_config, model=model, ref_model=None,
13                           tokenizer=tokenizer, dataset=dataset,
14                           data_collator=data_collator, optimizer=optimizer)
```

PPO Training

Then we can optimize the fine-tuned model with the **PPO algorithm** using the PPO trainer provided by the TRL library.

```
1 gen_kwargs = {
2     "top_k": 0.0,
3     "top_p": 1.0,
4     "do_sample": True,
5     "pad_token_id": tokenizer.pad_token_id,
6     "eos_token_id": tokenizer.eos_token_id
7 }
8 for batch in tqdm(ppo_trainer.dataloader):
9     responses_with_queries = ppo_trainer.generate(batch, **gen_kwargs)
10    # Compute rewards
11    _, _, values = rm_model(responses_with_queries)
12    rewards = values[-1]
13    # Run PPO step
14    ppo_trainer.step(queries, responses, rewards)
```

Open Source

We omit some details in the slides for clarity. The whole implementation is made available at:

<https://github.com/hiyouga/ChatGLM-Efficient-Tuning>

Our repository achieved 500+ stars and 50 forks within one month.

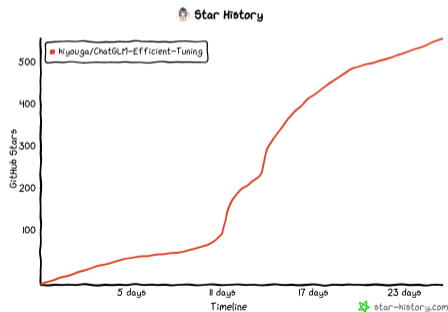


Table of Contents

1 Introduction

2 Preliminaries

3 Implementation

4 Experiment

Supervised Fine-Tuning Examples

We run an examples on the Chinese instruction-following data provided by [12]. The fine-tuned model achieves **better performance** compared with the base model.

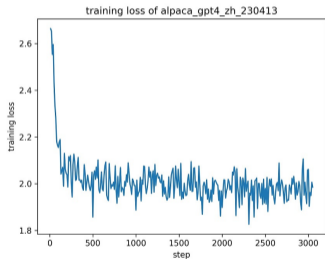


Figure: Training loss at the supervised fine-tuning.

	BLEU-4	Rouge-1	Rouge-2	Rouge-l
Base	15.75	34.51	15.11	26.18
SFT	17.01	36.77	16.83	28.86
	(+1.26)	(+2.26)	(+1.72)	(+2.68)

Table: Evaluation results.

RLHF Examples

We run an examples on comparison data for RLHF provided by [12]. The loss of reward model and the reward of the PPO model can be optimized during training.

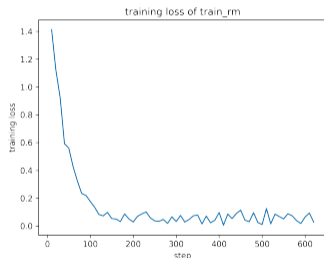


Figure: Training loss at reward model training.

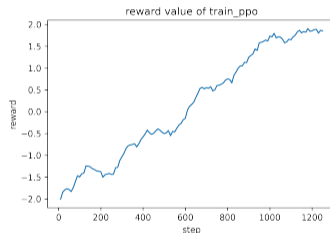







Figure: Reward value at PPO model training. We plot averaged value using window size=20.

References

-  Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.
Training language models to follow instructions with human feedback.
Advances in Neural Information Processing Systems, 35:27730–27744, 2022.
-  Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
Advances in neural information processing systems, 30, 2017.

-  Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
Bert: Pre-training of deep bidirectional transformers for language understanding.
In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, 2019.
-  Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov.
Roberta: A robustly optimized bert pretraining approach.
arXiv preprint arXiv:1907.11692, 2019.

References

-  Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al.
Improving language understanding by generative pre-training.
OpenAI blog, 2018.
-  Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le.
Xlnet: Generalized autoregressive pretraining for language understanding.
Advances in neural information processing systems, 32, 2019.
-  Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer.
Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

References

In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, 2020.

 Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu.

Exploring the limits of transfer learning with a unified text-to-text transformer.




Journal of Machine Learning Research, 21:1–67, 2020.

 Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano.

Learning to summarize with human feedback.

Advances in Neural Information Processing Systems, 33:3008–3021, 2020.

References

-  John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.
Proximal policy optimization algorithms.
arXiv preprint arXiv:1707.06347, 2017.
-  Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto.
Stanford alpaca: An instruction-following llama model.
https://github.com/tatsu-lab/stanford_alpaca, 2023.
-  Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao.
Instruction tuning with gpt-4.
arXiv preprint arXiv:2304.03277, 2023.