tatsu-lab / **stanford_alpaca** Public

Code and documentation to train Stanford's Alpaca models, and generate the data.

crfm.stanford.edu/alpaca/

Apache-2.0 license

14.7k stars **1.9k** forks

Star Watch

Code Issues 61 Pull requests 12 Actions Projects Security Ins

main

 **rtaori** Update README.md to clarify license ... 12 hours ago **28**

View code

README.md

Stanford Alpaca: An Instruction-following LLaMA Model

Code License Apache 2.0 Data License CC By NC 4.0 python 3.9+ code style black

This is the repo for the Stanford Alpaca project, which aims to build and share an instruction-following LLaMA model. The repo contains:

- The 52K data used for fine-tuning the model.
- The code for generating the data.
- The code for fine-tuning the model.

Note: We thank the community for feedback on Stanford-Alpaca and supporting our research. Our live demo is suspended until further notice.

Usage and License Notices: Alpaca is intended and licensed for research use only. The dataset is CC BY NC 4.0 (allowing only non-commercial use) and models trained using the dataset should not be used outside of research purposes.

Overview

The current Alpaca model is fine-tuned from a 7B LLaMA model [1] on 52K instruction-following data generated by the techniques in the Self-Instruct [2] paper, with some modifications that we discuss in the next section. In a preliminary human evaluation, we found that the Alpaca 7B model behaves similarly to the `text-davinci-003` model on the Self-Instruct instruction-following evaluation suite [2].

Alpaca is still under development, and there are many limitations that have to be addressed. Importantly, we have not yet fine-tuned the Alpaca model to be safe and harmless. We thus encourage users to be cautious when interacting with Alpaca, and to report any concerning behavior to help improve the safety and ethical considerations of the model.

Our initial release contains the data generation procedure, dataset, and training recipe. We intend to release the model weights if we are given permission to do so by the creators of LLaMA. For now, we have chosen to host a live demo to help readers better understand the capabilities and limits of Alpaca, as well as a way to help us better evaluate Alpaca's performance on a broader audience.

Please read our release blog post for more details about the model, our discussion of the potential harm and limitations of Alpaca models, and our thought process for releasing a reproducible model.

[1]: LLaMA: Open and Efficient Foundation Language Models. Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, Guillaume Lample. https://arxiv.org/abs/2302.13971v1

[2]: Self-Instruct: Aligning Language Model with Self Generated Instructions. Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, Hannaneh Hajishirzi. https://arxiv.org/abs/2212.10560

Data Release

`alpaca_data.json` contains 52K instruction-following data we used for fine-tuning the Alpaca model. This JSON file is a list of dictionaries, each dictionary contains the following fields:

- `instruction`: `str`, describes the task the model should perform. Each of the 52K instructions is unique.
- `input`: `str`, optional context or input for the task. For example, when the instruction is "Summarize the following article", the input is the article. Around 40% of the examples have an input.
- `output`: `str`, the answer to the instruction as generated by `text-davinci-003`.

We used the following prompts for fine-tuning the Alpaca model:

- for examples with a non-empty input field:

```
Below is an instruction that describes a task, paired with an input that
provides further context. Write a response that appropriately completes the
request.

### Instruction:
{instruction}

### Input:
{input}

### Response:
```

- for examples with an empty input field:

```
Below is an instruction that describes a task. Write a response that
appropriately completes the request.

### Instruction:
{instruction}

### Response:
```

During inference (eg for the web demo), we use the user instruction with an empty input field (second option).

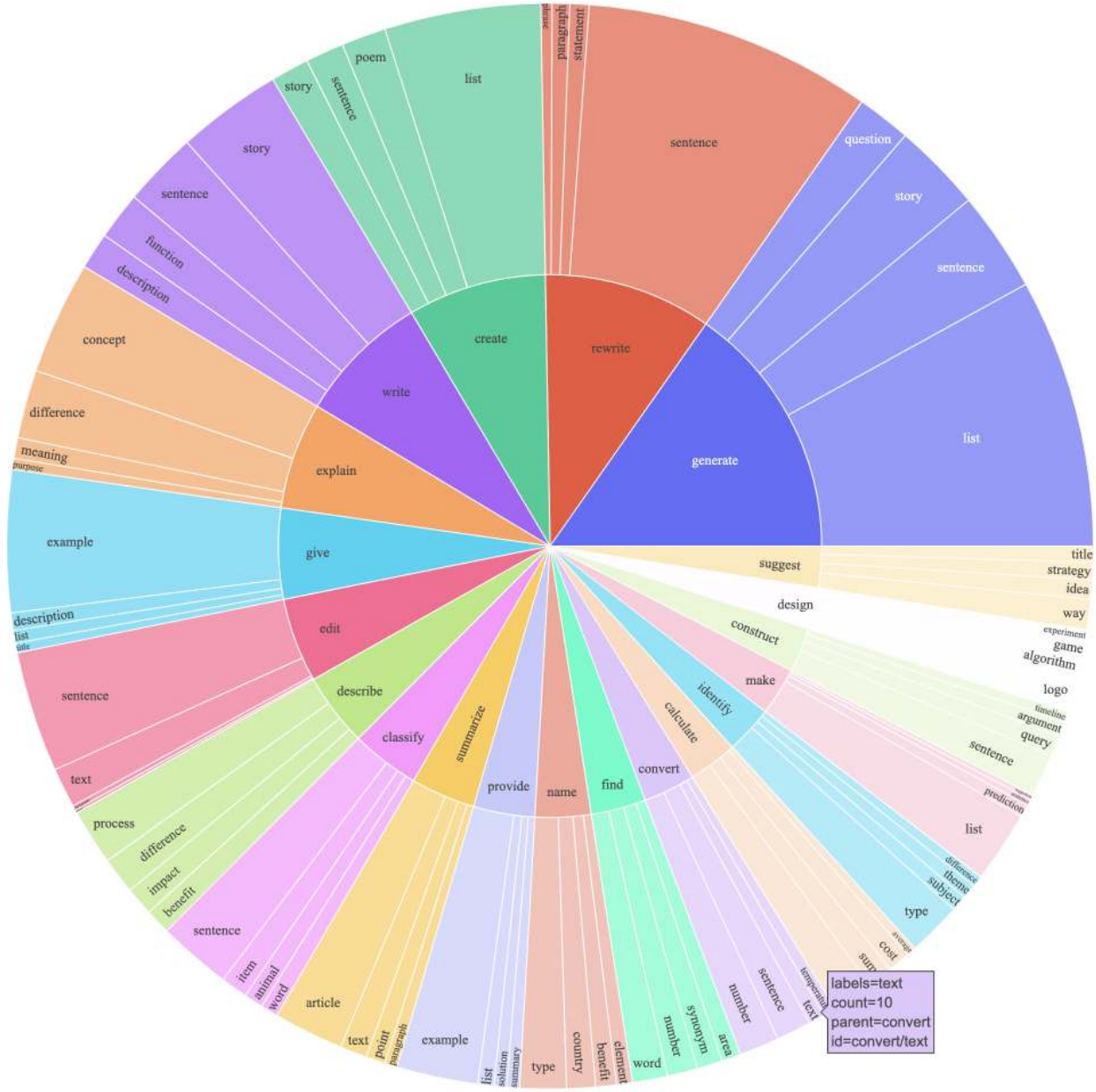
Data Generation Process

▶ **Running the code**

We built on the data generation pipeline from self-instruct and made the following modifications:

- We used `text-davinci-003` to generate the instruction data instead of `davinci`.
- We wrote a new prompt (`prompt.txt`) that explicitly gave the requirement of instruction generation to `text-davinci-003`. Note: there is a slight error in the prompt we used, and future users should incorporate the edit in #24
- We adopted much more aggressive batch decoding, i.e., generating 20 instructions at once, which significantly reduced the cost of data generation.
- We simplified the data generation pipeline by discarding the difference between classification and non-classification instructions.
- We only generated a single instance for each instruction, instead of 2 to 3 instances as in [1].

This produced an instruction-following dataset with 52K examples obtained at a much lower cost (less than $500). In a preliminary study, we also find our 52K generated data to be much more diverse than the data released by self-instruct. We plot the below figure (in the style of Figure 2 in the self-instruct paper to demonstrate the diversity of our data. The inner circle of the plot represents the root verb of the instructions, and the outer circle represents the direct objects.



Fine-tuning

We fine-tune our models using standard Hugging Face training code with the following hyperparameters:

Hyperparameter	Value
Batch size	128
Learning rate	2e-5
Epochs	3
Max length	512

Hyperparameter	Value
Weight decay	0

Given Hugging Face hasn't officially supported the LLaMA models, we fine-tuned LLaMA with Hugging Face's transformers library by installing it from a particular fork (i.e. this PR to be merged). The hash of the specific commit we installed was `68d640f7c368bcaaaecfc678f11908ebbd3d6176`.

To reproduce our fine-tuning runs for LLaMA, first install the requirements

```
pip install -r requirements.txt
```

Then, install the particular fork of Hugging Face's transformers library.

Below is a command that fine-tunes LLaMA-7B with our dataset on a machine with 4 A100 80G GPUs in FSDP `full_shard` mode. We were able to reproduce a model of similar quality as the one we hosted in our demo with the following command using **Python 3.10**. Replace `<your_random_port>` with a port of your own, `<your_path_to_hf_converted_llama_ckpt_and_tokenizer>` with the path to your converted checkpoint and tokenizer (following instructions in the PR), and `<your_output_dir>` with where you want to store your outputs.

```
torchrun --nproc_per_node=4 --master_port=<your_random_port> train.py \
    --model_name_or_path <your_path_to_hf_converted_llama_ckpt_and_tokenizer>
    --data_path ./alpaca_data.json \
    --bf16 True \
    --output_dir <your_output_dir> \
    --num_train_epochs 3 \
    --per_device_train_batch_size 4 \
    --per_device_eval_batch_size 4 \
    --gradient_accumulation_steps 8 \
    --evaluation_strategy "no" \
    --save_strategy "steps" \
    --save_steps 2000 \
    --save_total_limit 1 \
    --learning_rate 2e-5 \
    --weight_decay 0. \
    --warmup_ratio 0.03 \
    --lr_scheduler_type "cosine" \
    --logging_steps 1 \
    --fsdp "full_shard auto_wrap" \
```

```
    --fsdp_transformer_layer_cls_to_wrap 'LLaMADecoderLayer' \
    --tf32 True
```

Warning

`fsdp_transformer_layer_cls_to_wrap` must be set to the name of the specific decoder layer. The LLaMA Hugging Face PR is not stable. Earlier commits used the name `LLaMADecoderLayer` for their decoder layer (the commit hash our code is based on this). More recent commits use `LlamaDecoderLayer` (notice the small case difference). Not setting `fsdp_transformer_layer_cls_to_wrap` to the correct name will lead to drastic slowdowns in training.

Side notes

The same script also works for OPT fine-tuning. Here's an example for fine-tuning OPT-6.7B

```
torchrun --nproc_per_node=4 --master_port=<your_random_port> train.py \
    --model_name_or_path "facebook/opt-6.7b" \
    --data_path ./alpaca_data.json \
    --bf16 True \
    --output_dir <your_output_dir> \
    --num_train_epochs 3 \
    --per_device_train_batch_size 4 \
    --per_device_eval_batch_size 4 \
    --gradient_accumulation_steps 8 \
    --evaluation_strategy "no" \
    --save_strategy "steps" \
    --save_steps 2000 \
    --save_total_limit 1 \
    --learning_rate 2e-5 \
    --weight_decay 0. \
    --warmup_ratio 0.03 \
    --lr_scheduler_type "cosine" \
    --logging_steps 1 \
    --fsdp "full_shard auto_wrap" \
    --fsdp_transformer_layer_cls_to_wrap 'OPTDecoderLayer' \
    --tf32 True
```

Note the given training script is meant to be simple and easy to use, and is not particularly optimized. To run on more gpus, you may prefer to turn down `gradient_accumulation_steps` to keep a global batch size of 128. Global batch size has not been tested for optimality.

Authors

All grad students below contributed equally and the order is determined by random draw.

- Rohan Taori
- Ishaan Gulrajani
- Tianyi Zhang
- Yann Dubois
- Xuechen Li

All advised by Tatsunori B. Hashimoto. Yann is also advised by Percy Liang and Xuechen is also advised by Carlos Guestrin.

Citation

Please cite the repo if you use the data or code in this repo.

```
@misc{alpaca,
  author = {Rohan Taori and Ishaan Gulrajani and Tianyi Zhang and Yann
Dubois and Xuechen Li and Carlos Guestrin and Percy Liang and Tatsunori B.
Hashimoto },
  title = {Stanford Alpaca: An Instruction-following LLaMA model},
  year = {2023},
  publisher = {GitHub},
  journal = {GitHub repository},
  howpublished = {\url{https://github.com/tatsu-lab/stanford_alpaca}},
}
```

Naturally, you should also cite the original LLaMA paper [1] and the Self-Instruct paper [2].

Acknowledgements

We thank Yizhong Wang for his help in explaining the data generation pipeline in Self-Instruct and providing the code for the parse analysis plot. We thank Yifan Mai for helpful support, and members of the Stanford NLP Group as well as the Center for Research on Foundation Models (CRFM) for their helpful feedback.

Releases

No releases published

Packages

No packages published

Contributors 7

Languages

- **Python** 100.0%