SplitLoRA: A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models

SplitLoRA is the first SL LLM fine-tuning framework. SplitLoRA is built on the split federated learning (SFL) framework, amalgamating the advantages of parallel training from FL and model splitting from
SL, thus greatly enhancing the training efficiency. It is worth noting that SplitLoRA is the inaugural open-source benchmark for SL LLM fine-tuning, providing a foundation for research efforts dedicated to advancing SL LLM fine-tuning. The project page is available at https://fdu-inc.github.io/splitlora/ and technical report can be found at https://arxiv.org/pdf/2407.00952

Citation

```
@inproceedings{
  @article{lin2024splitlora,
  title={{SplitLoRA: A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models}},
  author={Lin, Zheng and Hu, Xuanjie and Zhang, Yuxin and Chen, Zhe and Fang, Zihan and Chen, Xianhao and Li, Ang and Vepakomma, Praneeth and Gao, Yue},
  journal={arXiv preprint arXiv:2407.00952},
  year={2024}
  }
}
```

User Guide

This repository is based on LoRA.

1 Introduction

SplitLoRA contains the source code of the Python package loralib and a example of how to integrate it with PyTorch models, GPT2-s. We only support PyTorch for now.In the future, we will integrate more open source LLMs and more tasks into the SplitLoRA framework

- The source code of the Python package loralib
- LoRA fine-tuning implementation of large language models
- LoRA fine-tuning implementation of large language model under `SplitLoRA framework`

2 Build

2.1 Environment Requirements

We have verified in the environment below:

- OS: Ubuntu 18.04
- Python: 3.7.16

	torch 1.7.1+cu101	transformers 3.3.1	spacy	tqdm	tensorboard	progress

Note: You still need the original pre-trained checkpoint from [Hugging Face](#) to use the LoRA checkpoints.

2.2 Quick Start

1. Installing `loralib` is simply

```
pip install loralib
# Alternatively
# pip install git+https://github.com/microsoft/LoRA
```

2. You can choose to adapt some layers by replacing them with counterparts implemented in `loralib`. We only support `nn.Linear`, `nn.Embedding`, and `nn.Conv2d` for now. We also support a `MergedLinear` for cases where a single `nn.Linear` represents more than one layers, such as in some implementations of the attention `qkv` projection (see Additional Notes for more).

```
# ===== Before =====
# layer = nn.Linear(in_features, out_features)

# ===== After ======
import loralib as lora
# Add a pair of low-rank adaptation matrices with rank r=16
layer = lora.Linear(in_features, out_features, r=16)
```

3. Before the training loop begins, mark only LoRA parameters as trainable.

```
import loralib as lora
model = BigModel()
# This sets requires_grad to False for all parameters without the string "lora_"
in their names
lora.mark_only_lora_as_trainable(model)
# Training loop
for batch in dataloader:
   ...
```

4. When saving a checkpoint, generate a `state_dict` that only contains LoRA parameters.

```
# ===== Before =====
# torch.save(model.state_dict(), checkpoint_path)
# ===== After =====
torch.save(lora.lora_state_dict(model), checkpoint_path)
```

5. When loading a checkpoint using `load_state_dict`, be sure to set `strict=False`.

```
# Load the pretrained checkpoint first
model.load_state_dict(torch.load('ckpt_pretrained.pt'), strict=False)
# Then load the LoRA checkpoint
model.load_state_dict(torch.load('ckpt_lora.pt'), strict=False)
```

2.3 Steps To Reproduce Our Results

1. You can start with the following docker image: `nvcr.io/nvidia/pytorch:20.03-py3` on a GPU-capable machine, but any generic PyTorch image should work.

```
docker run -it nvcr.io/nvidia/pytorch:20.03-py3
```

2. Clone the repo and install dependencies in a virtual environment (remove sudo if running in docker container):

```
sudo apt-get update
sudo apt-get -y install git jq virtualenv
git clone https://github.com/microsoft/LoRA.git; cd LoRA
virtualenv -p `which python3` ./venv
. ./venv/bin/activate
pip install -r requirement.txt
bash download_pretrained_checkpoints.sh
bash create_datasets.sh
cd ./eval
bash download_evalscript.sh
cd ..
```

Now we are ready to replicate the results

3 SplitLoRA Module Libraries

3.1 Repository

Our implementation is based on the fine-tuning code for GPT-2 in Hugging Face.
There are several directories in this repo:

- src/ contains the source code used for data processing, training, and decoding.
- eval/ contains the code for task-specific evaluation scripts.
- data/ contains the raw data we used in our experiments.
- vocab/ contains the GPT-2 vocabulary files.

3.2 Hyper-Parameter

```
 --nproc_per_node=1: Specifies the number of processes per node, set to 1 here.

--train_data: Specifies the path to the training data, set to
./data/e2e/train0.jsonl,train1.jsonl,train2.jsonl.

--valid_data: Specifies the path to the validation data, set to
./data/e2e/valid.jsonl.

--train_batch_size: Specifies the training batch size, set to 8.

--grad_acc: Specifies the number of gradient accumulation steps, set to 1, which
means the gradient is updated once per batch.

--valid_batch_size: Specifies the validation batch size, set to 4.

--seq_len: Specifies the sequence length, set to 512.

--model_card: Specifies the path to the model configuration file, set to gpt2.md.

--init_checkpoint: Specifies the path to the initial checkpoint file for model
initialization, set to ./pretrained_checkpoints/gpt2-pytorch_model.bin.

--platform: Specifies the execution platform, set to local.

--clip: Specifies the threshold for gradient clipping, set to 0.0, which means no
gradient clipping is performed.

--lr: Specifies the learning rate, set to 0.0002.

--weight_decay: Specifies the weight decay (L2 regularization) parameter, set to
0.01.

--correct_bias: Specifies whether to correct biases, default is False.

--adam_beta2: Specifies the beta2 parameter for the Adam optimizer, set to 0.999.

--scheduler: Specifies the type of learning rate scheduler, set to linear.

--warmup_step: Specifies the number of warm-up steps for linear learning rate
warm-up, set to 500.

--max_epoch: Specifies the maximum number of training epochs, set to 5.

--save_interval: Specifies the interval steps for model saving, set to 1000.

--lora_dim: Specifies the dimension of LoRA (Local-Regional Attention), set to 4.

--lora_alpha: Specifies the alpha hyperparameter for LoRA, set to 32.

--lora_dropout: Specifies the dropout rate for LoRA, set to 0.1.

--label_smooth: Specifies the label smoothing parameter, set to 0.1.
```

```
--work_dir: Specifies the working directory where the models and log files are
saved, set to ./trained_models/GPT2_S/e2e.

--random_seed: Specifies the random seed, set to 110.
```

4 Training Process

1. Train GPT-2 Medium with SplitLoRA

At examples/NLG, run:

```
python -m torch.distributed.launch --nproc_per_node=1 --use_env
src/gpt2_ft_sfl.py \
--train_data0 ./data/e2e/train0.jsonl \
--train_data1 ./data/e2e/train1.jsonl \
--train_data2 ./data/e2e/train2.jsonl \
--valid_data ./data/e2e/valid.jsonl \
--train_batch_size 4 \
--grad_acc 1 \
--valid_batch_size 4 \
--seq_len 512 \
--model_card gpt2.md \
--init_checkpoint ./pretrained_checkpoints/gpt2-medium-pytorch_model.bin \
--platform local \
--clip 0.0 \
--lr 0.0002 \
--weight_decay 0.01 \
--correct_bias \
--adam_beta2 0.999 \
--scheduler linear \
--warmup_step 500 \
--max_epoch 5 \
--save_interval 400000 \
--lora_dim 2 \
--lora_alpha 32 \
--lora_dropout 0.1 \
--label_smooth 0.1 \
--work_dir ./trained_models/GPT2_M/e2e \
--random_seed 40
```

2. Generate outputs from the trained model using beam search:

```
python -m torch.distributed.launch --nproc_per_node=1 src/gpt2_beam.py \
    --data ./data/e2e/test.jsonl \
    --batch_size 1 \
    --seq_len 512 \
    --eval_len 64 \
    --model_card gpt2.md \
    --init_checkpoint ./trained_models/GPT2_S/e2e/{model.name.pt} \
    --platform local \
    --lora_dim 4 \
    --lora_alpha 32 \
    --beam 10 \
    --length_penalty 0.8 \
    --no_repeat_ngram_size 4 \
```

```
    --repetition_penalty 1.0 \
    --eos_token_id 628 \
    --work_dir ./trained_models/GPT2_S/e2e \
    --output_file predict.26289.b10p08r4.jsonl
```

3. Decode outputs from step (2)

```
python src/gpt2_decode.py \
    --vocab ./vocab \
    --sample_file ./trained_models/GPT2_M/e2e/predict.26289.b10p08r4.jsonl \
    --input_file ./data/e2e/test_formatted.jsonl \
    --output_ref_file e2e_ref.txt \
    --output_pred_file e2e_pred.txt
```

4. Run evaluation on E2E test set

```
python eval/e2e/measure_scores.py e2e_ref.txt e2e_pred.txt -p
```

5. Citation

```
@inproceedings{
  @article{lin2024splitlora,
  title={{SplitLoRA: A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models}},
  author={Lin, Zheng and Hu, Xuanjie and Zhang, Yuxin and Chen, Zhe and Fang, Zihan and Chen, Xianhao and Li, Ang and Vepakomma, Praneeth and Gao, Yue},
  journal={arXiv preprint arXiv:2407.00952},
  year={2024}
  }
}
```

Appendix

If you want to know more detailed information about Lora, see https://github.com/microsoft/LoRA.

If you've found SplitLoRA framework useful for your project, please cite our paper.

Contact Us

- Mr. Xuanjie Hu, email: 23210240184@m.fudan.edu.cn