

Automatic speech recognition

Peft(parameter efficient fine-tuning):

LoRa(low-rank adaptation of Large language models):

We propose Low-Rank Adaptation, or LoRA, which freezes the **pre-trained model weights** and **injects trainable rank decomposition matrices into each layer of the Transformer architecture**, greatly **reducing** the number of **trainable parameters for downstream tasks**.

Compared to GPT-3 175B, which is fine-tuned with Adam, LoRA can reduce the number of trainable parameters by **10,000 times** and the GPU memory requirement by 3 times.

LoRA performs on par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having **fewer trainable parameters**, a higher training throughput, and, unlike adapters, no additional inference latency. We also provide an empirical investigation into rank deficiency in language model adaptation, which sheds light on the efficacy of LoRA.

We release a package that facilitates the integration of LoRA with PyTorch models and provides our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at [this https URL](https://github.com/microsoft/LoRA).

LoRA reduces the number of trainable parameters by learning pairs of rank-decomposition matrices while freezing the original weights.

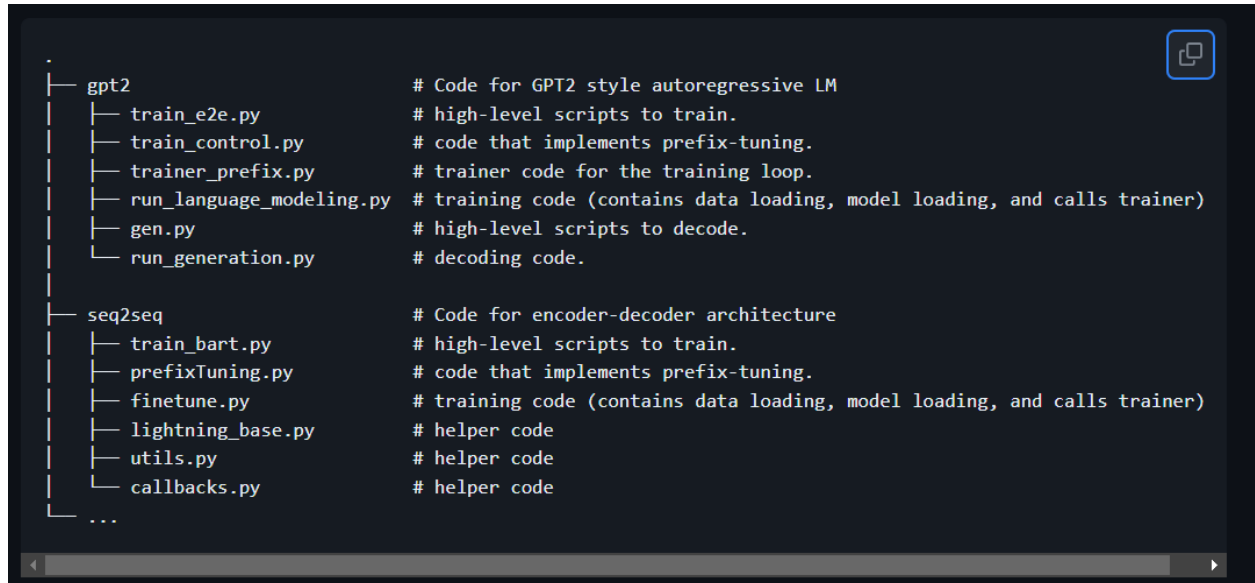
This vastly reduces the storage requirement for large language models adapted to specific tasks and enables efficient task-switching during deployment all without introducing inference latency. LoRA also outperforms several other adaptation methods including adapter, prefix-tuning, and fine-tuning.

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).

Prefix Tuning:

Prefix tuning on GPT2 style autoregressive LM

Prefix tuning on bart encoder and decoder architectures.



```
.
├── gpt2                                # Code for GPT2 style autoregressive LM
│   ├── train_e2e.py                  # high-level scripts to train.
│   ├── train_control.py             # code that implements prefix-tuning.
│   ├── trainer_prefix.py            # trainer code for the training loop.
│   ├── run_language_modeling.py     # training code (contains data loading, model loading, and calls trainer)
│   ├── gen.py                       # high-level scripts to decode.
│   └── run_generation.py            # decoding code.
├── seq2seq                           # Code for encoder-decoder architecture
│   ├── train_bart.py                # high-level scripts to train.
│   ├── prefixTuning.py              # code that implements prefix-tuning.
│   ├── finetune.py                  # training code (contains data loading, model loading, and calls trainer)
│   ├── lightning_base.py            # helper code
│   ├── utils.py                     # helper code
│   └── callbacks.py                 # helper code
└── ...
```

Prompt tuning:

The power of Scale for parameter-efficient prompt tuning:

The code reproduces the experiment

They are built on **T5X** which defines the model and training loop.

Flaxformer: defines the actual model computation.

Flax: defines the low-level model layers.

Jax: provides the actual execution.

a simple yet effective mechanism for learning “**soft prompts**” to condition frozen language models to perform specific downstream tasks. Unlike the discrete text

prompts used by GPT-3, **soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples**. Our end-to-end learned approach outperforms GPT-3's few-shot learning by a large margin. More remarkably, through ablations on model size using T5, we show that prompt tuning becomes more competitive with scale: as models exceed billions of parameters, our method "closes the gap" and matches the strong performance of model tuning (where all model weights are tuned). This finding is especially relevant because large models are costly to share and serve and the ability to reuse one frozen model for multiple downstream tasks can ease this burden. **Our method can be seen as a simplification of the recently proposed "prefix tuning" of Li and Liang (2021) and we provide a comparison to this and other similar approaches**. Finally, we show that conditioning a frozen model with soft prompts confers benefits in robustness to domain transfer and enables efficient "prompt ensembling." We release code and model checkpoints to reproduce our experiments.

Using Pytorch:

Implementation of soft embeddings from <https://arxiv.org/abs/2104.08691v1> using Pytorch and Huggingface transformers.

Using nn.Embedding creates a look up tables.

1.2. Soft-Prompts

Soft-prompts are represented as a parameter P_e . The prompt is then concatenated to the embedded input forming a single matrix $[P_e, X_e]$ which then flows through the encoder-decoder as normal.

Prompt tuning becomes more competitive with model tuning as scale increases. At the XXL size (11 billion parameters), prompt tuning matches even the stronger multi-task model tuning baseline, despite having over 20,000 times fewer task-specific parameters.

LOHA

FedPara: Low-Rank Hadamard Product for Communication-Efficient Federated Learning

communication-efficient parameterization, FedPara, for federated learning (FL) to overcome the burdens on frequent model uploads and downloads. Our method **re-parameterizes weight parameters of layers using low-rank weights followed by the Hadamard product**. Compared to the conventional low-rank parameterization, our FedPara method is not restricted to low-rank constraints, and thereby it has a far larger capacity. This property enables to achieve comparable performance while requiring 3 to 10 times lower communication costs than the model with the original layers, which is not achievable by the traditional low-rank methods. The efficiency of our method can be further improved by combining with other efficient FL optimizers. In addition, we extend our method to a personalized FL application, pFedPara, which separates parameters into global and local ones. We show that pFedPara outperforms competing personalized FL methods with more than three times fewer parameters.

nn.Parameter matrix structure