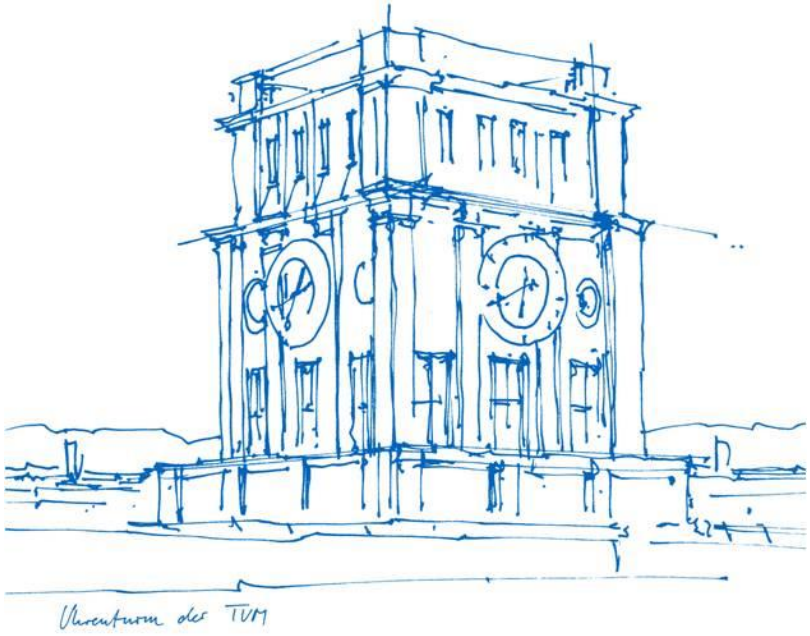
SELECTIVE PARAMETER UPDATING - MEETING 5



Coşku Barış Coşlu

Mato Gudelj

Baykam Say

28 Jun 2023

Contents

Plan from last meeting…

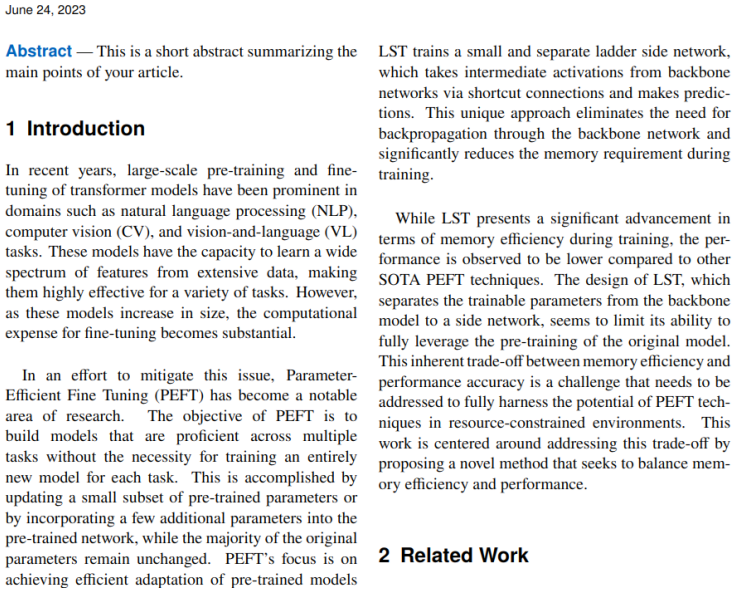
- Implement k-ladder LST and record some results
- Implement MeZO + LST
- Implement Side LoRA
- Start working on the report
- Brainstorm more approaches
- …keep reading literature

Plan from last meeting…

- Implement k-ladder LST and record some results
- Implement MeZO + LST
- Implement Side LoRA
- Start working on the report
- Brainstorm more approaches
- …keep reading literature

Report

- Started writing the report
- Mostly introduction, outline, etc.
  - Not clear which method we will center it around
  - Multiple methods possible
    - Side LoRA
    - k-LST
    - MeZO LST…



June 24, 2023

Abstract — This is a short abstract summarizing the main points of your article.

1 Introduction

In recent years, large-scale pre-training and fine-tuning of transformer models have been prominent in domains such as natural language processing (NLP), computer vision (CV), and vision-and-language (VL) tasks. These models have the capacity to learn a wide spectrum of features from extensive data, making them highly effective for a variety of tasks. However, as these models increase in size, the computational expense for fine-tuning becomes substantial.

In an effort to mitigate this issue, Parameter-Efficient Fine Tuning (PEFT) has become a notable area of research. The objective of PEFT is to build models that are proficient across multiple tasks without the necessity for training an entirely new model for each task. This is accomplished by updating a small subset of pre-trained parameters or by incorporating a few additional parameters into the pre-trained network, while the majority of the original parameters remain unchanged. PEFT's focus is on achieving efficient adaptation of pre-trained models

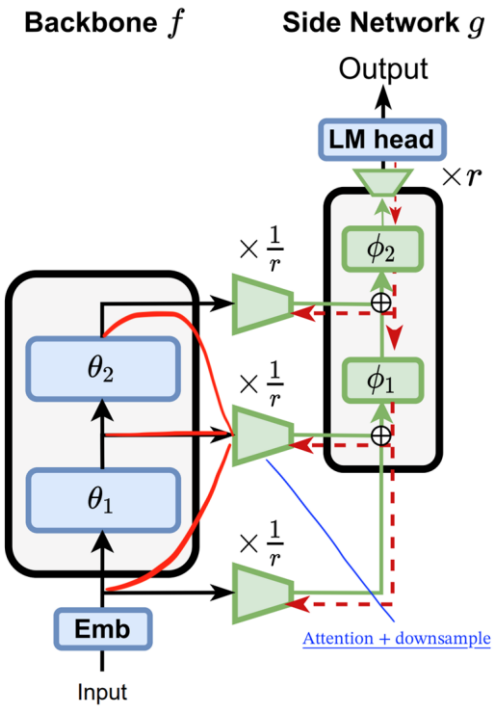
LST trains a small and separate ladder side network, which takes intermediate activations from backbone networks via shortcut connections and makes predictions. This unique approach eliminates the need for backpropagation through the backbone network and significantly reduces the memory requirement during training.

While LST presents a significant advancement in terms of memory efficiency during training, the performance is observed to be lower compared to other SOTA PEFT techniques. The design of LST, which separates the trainable parameters from the backbone model to a side network, seems to limit its ability to fully leverage the pre-training of the original model. This inherent trade-off between memory efficiency and performance accuracy is a challenge that needs to be addressed to fully harness the potential of PEFT techniques in resource-constrained environments. This work is centered around addressing this trade-off by proposing a novel method that seeks to balance memory efficiency and performance.

2 Related Work

Recap: k-LST idea

- Currently: i-th backbone output used as i-th ladder input
- Idea: compute input from fixed k length window of backbone outputs
  - LST [1] is a special case for k=1
  - Mix/combine with attention
    - Positional encoding for blocks



k-LST results

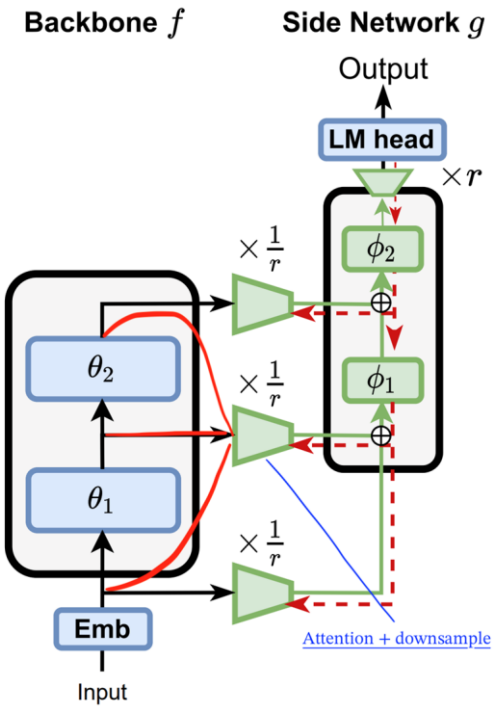
- DistilBERT didn’t show any difference
  - Only 6 layers
  - Little benefit from combining the features?
  - => Try with RoBERTa-large (24 layers)
- A lot of hyperparameter tuning left, both for k-LST and baseline LST
- k-LST doesn’t add a lot of overhead (9-LST about 10% slower, a bit more VRAM, both around 3 GB)

Method	SST-2 Accuracy	SST-2 F1
RoBERTa-large + LST	93.12%	93.10%
RoBERTa-large + 9-LST	94.50%	94.62%
RoBERTa-large + Full FT	96.40%	-

Additional k-LST ideas

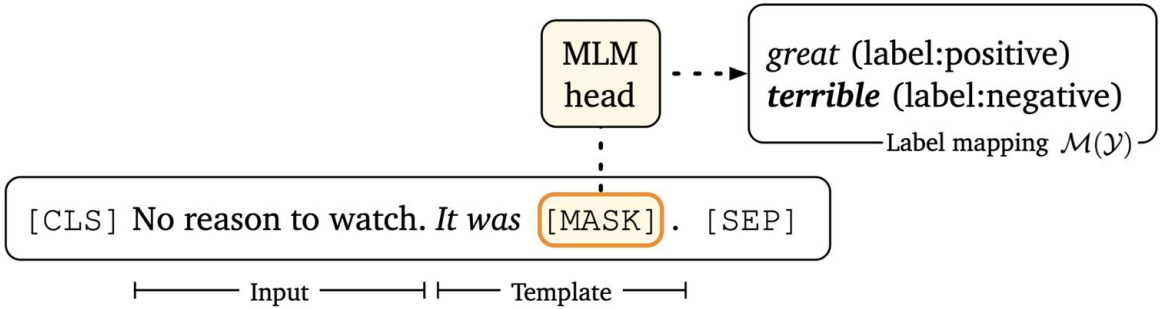
- Observation: Larger “k” in k-LST more easily overfits
- Idea: Add dropout to ladder connections

- Observation: k-LST still lags behind other methods in F1/Acc
  - Larger reduction improves results, likely due to less overfitting
  - We are randomly initializing a lot of parameters
- Idea: Pretrain LST before FT
- Other things:
  - Try unfreezing the last backbone layer or two
    - VRAM usage still very low, but might claw back missing perf
  - Try different downsampling strategies
    - Currently it’s “consistent” -> each feature is always downsampled by the same downsampling block
  - Operating on backbone features costs nothing -> Pool all feats?



MeZO

- Estimate gradients with only forward passes: Very memory-efficient
- Drawback: Significantly increases training time requirement
- Requires prompt-based fine-tuning:
  - Apply a template with a mask to input sentences
  - Pick label words for positive and negative labels
  - Classify based on probabilities of label words on the mask



MeZO Results

Method	SST-2 Accuracy	SST-2 F1
DistilBERT Full fine-tuning	91.28%	91.52%
DistilBERT MeZO	82.57%	82.69%

Method	SST-2 Accuracy	SST-2 F1
RoBERTa Full fine-tuning*	96.4%	-
RoBERTa MeZO	89.79%	89.97%

* taken from their own papers

MeZO + LST

Approach:

1. Fine-tune masked LM using MeZO
2. Load model from checkpoint
3. Fine-tune model using LST
    - As a masked LM (continue with prompt-based FT)
    - As a sequence classifier (with classification head)
        - Faster, uses less VRAM, achieves better accuracy

MeZO + LST Results

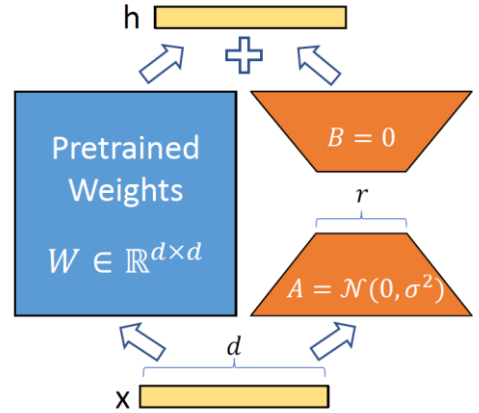
Method	SST-2 Accuracy / F1
DistilBERT Full fine-tuning	91.28% / 91.52%
DistilBERT MeZO	82.57% / 82.69%
DistilBERT LST	88.07% / 88.37%
DistilBERT MeZO + LST	86.24% / 86.67%

Method	SST-2 Accuracy / F1
RoBERTa Full fine-tuning*	96.4% / -
RoBERTa MeZO	89.79% / 89.97%
RoBERTa LST	92.54% / 92.83%
RoBERTa MeZO + LST	92.89% / 92.84%

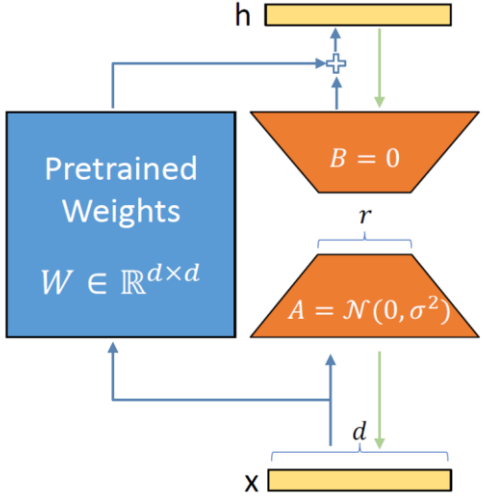
- Barely any improvement is observed over standard LST.
- Could still be interesting to experiment with hyperparameters and try with k-LST

Side-LoRA

- LoRA:
  - Pretrained weights are kept as frozen trainable parameters
  - During backprop gradients of the pretrained weights are still calculated if we need to train previous layers (x in diagram)
- Problem: unnecessary gradient calculations during backprop
- Solution: Side-LoRA
  - Keep pretrained weights as buffers
  - During backprop only go through the newly inserted LoRA layers (not the pretrained weights)



LoRA [3]



Side-LoRA

Side-LoRA Results

- DistilBERT
  - 64% reduction in the number of parameters during gradient calculation compared to LoRA (LoRA - 66M, Side-LoRA - 23M)
  - 11% reduction in total training runtime compared to LoRA (Lora - 32 minutes, Side-LoRA - 28 minutes)
- Virtually no performance loss compared to LoRA during fine tuning
- No additional memory reduction compared to LoRA

Side-LoRA Results

Method	Model	Accuracy	F1 Score
Full FT	DistilBERT	90.59%	-
LoRA	DistilBERT	90.02%	90.24%
Side-LoRA	DistilBERT	90.25%	90.48%
Full FT* [4]	RoBERTa Large	96.40%	-
LoRA* [3]	RoBERTa Large	96.20% (+/- 0.5%)	-
Side-LoRA	RoBERTa Large	96.10%	96.19%

* taken from their own papers

Open Issues & Solutions

- Issues
  - Need to do a lot of hyperparameter tuning
  - GPU heats up the room
  - Multiple potential methods
- Solutions
  - Run experiments 24/7
  - Decide on the method after exhaustive hyperparameter tuning
    - Alternatively: Include all methods that show improvement

Plan for the Next Two Weeks

- Do exhaustive hyperparameter tuning
- Implement k-LST ideas
- Start assembling main results for poster and report
- Brainstorm more approaches
- …keep reading literature

References

1. Yi-Lin Sung, Jaemin Cho, Mohit Bansal: “LST: Ladder Side-Tuning for Parameter and Memory Efficient Transfer Learning”, 2022; arXiv:2206.06522
2. Malladi, Sadhika, et al. ‘Fine-Tuning Language Models with Just Forward Passes’. *arXiv [Cs.LG]*, 2023; arXiv:2305.17333
3. Hu, Edward J., et al. ‘LoRA: Low-Rank Adaptation of Large Language Models’. ArXiv [Cs.CL], 2021; arXiv:2106.09685
4. Liu, Yinhan, et al. ‘RoBERTa: A Robustly Optimized BERT Pretraining Approach’. ArXiv [Cs.CL], 2019; arXiv:1907.11692