

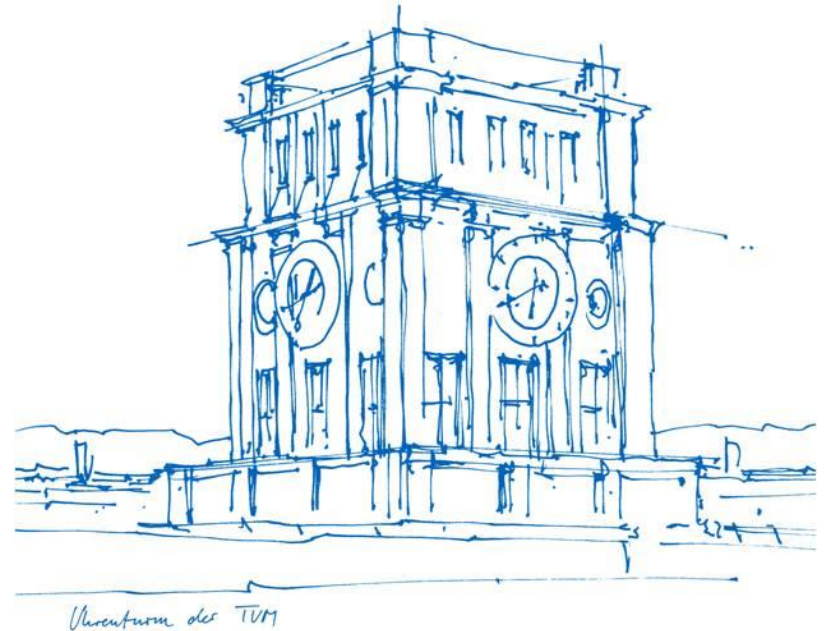
# SELECTIVE PARAMETER UPDATING - MEETING 6

Coşku Barış Coşlu

Mato Gudelj

Baykam Say

12 Jul 2023



# Contents

1. Plan from last meeting
2. Poster
3. k-LST
4. k-LST + Ladder connection dropout
5. MeZO + LST
6. MeZO + Weight Update Skipping
7. SiVA
8. Open Issues & Solutions
9. Plan for the Next Two Weeks

## Plan from last meeting...

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

# Poster

- Began writing text
  - 'Motivation' is complete
  - Intro to 'Results' is complete
- Need to prepare visuals
- Keep or change the layout of the template?

Please adjust group name in tumuser.shy.  
Department of Electrical and Computer Engineering  
Technical University of Munich

## Poster title

### Subtitle of the poster

First Author<sup>1</sup>, Second Author<sup>2</sup>, and Last Author<sup>1</sup>

#### Motivation

As the model sizes grow from billions to trillions of parameters for state-of-the-art language models such as GPT-4, the already resource-intensive task of training them has become impossible for but the largest and wealthiest private organizations. Even fine-tuning these models for specific tasks necessitates substantial computational power, memory, and time, making the process infeasible for smaller institutions and individuals. Furthermore, the excessive computational resources required for training and fine-tuning these models raise environmental concerns due to the associated energy consumption. Selectively updating parameters during training can provide a viable solution to these challenges. By optimizing the number of parameters required for fine-tuning, we can significantly reduce the computational resources required, making the process more affordable and accessible to a wider range of organizations and researchers. While current parameter-efficient fine-tuning (PEFT) methods have made significant progress in mitigating the challenges associated with fine-tuning large language models, there is still ample room for improvement. As the area of research is relatively new, fine-tuning techniques such as adapters, prompt tuning, LST, and LoRA can introduce additional parameters to the model, limit the expressiveness of the model by not updating enough of the parameters, and harm the quality of the model for increased efficiency. By improving on these PEFT methods, we aim to reduce the computational overhead of fine-tuning without compromising the quality of the model, thereby reducing the carbon footprint and allowing smaller organizations to utilize modern LLMs better.

#### Results

We have made improvements over each of the parameter-efficient fine-tuning methods we have focused on. As a result, we present a new PEFT method that can be used depending on the desired benefits during fine-tuning. InLoRA can be used to achieve the lowest possible memory usage, InLST can be used to achieve a balanced reduction in memory usage and computation time with minimal loss of model performance, and InLoRA can be used to achieve the highest possible reduction in memory usage and computation time when no compromise can be made on model performance.



Figure 1 The figure caption.

#### Method and Model

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam eros lacinia, convallis eget, conestatur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Maecenas ut leo. Cras viverra metus rhoncus quam. Itaque eu tulla vestibulum urna fringilla ultrices. Phasellus eu tellus ut amet tortor gravida placerat. Integer sapien est, laoreet in, pretium tui, viverra ac, nunc. Phasellus eget eros vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, metus ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci ut amet and dignism vulput.

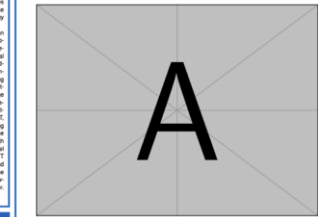


Figure 2 This is a figure caption.

Nam id ligula, fringilla a, accumsan sed, vestibulum ut, nisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, laboris vitae, ultrices et, tellus. Donec aliquet, tortor sed accumsan bibendum, nisi ligula aliquet magna, vel dui eros orci metus id. Morbi ac orci et neque tristique. Quamquam et malesuada. Cras nec ante. Pellentesque a nulla. Cum semis vulputate periculis et magna de parturient montes, nascetur ridiculus mus. Aliquam risus elit nulla. Nulla ultricies vestibulum turpis. Pellentesque varius varius mauris. Nulla malesuada porttitor diam. Donec felis enim, congue non, vulputate et, tristique tristique, felis. Vivamus viverra fermentum felis. Donec congue pellentesque enim. Phasellus adipiscing semper nisi. Nam tunc tunc massa ac quam. Sed diam turpis, convallis vitae, placerat a, risus nec, leo. Maecenas lacinia. Nam ipsum ligula, vestibulum at, accumsan nec, congue a, quam. Nam blandit ligula fringilla magna. Nam sed lectus congue lectus. Sed turpis nulla vitae enim. Pellentesque tristique purus vel magna. Integer non enim. Praesent auctor nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

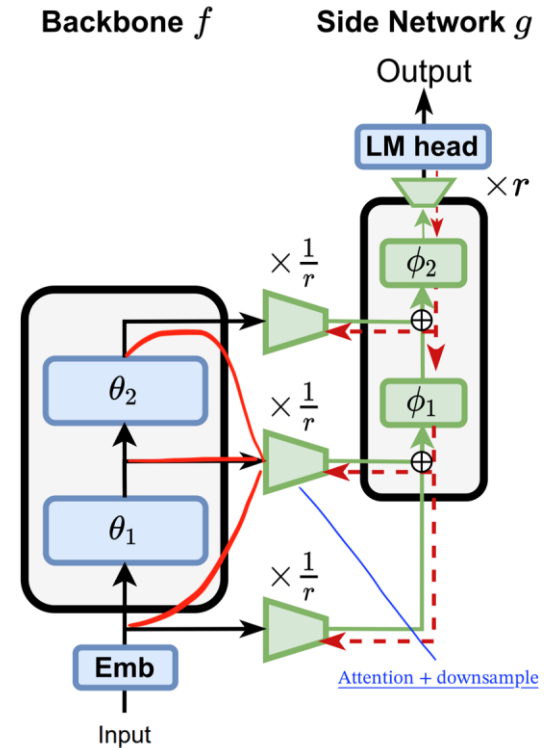
# k-LST

Tried unfreezing the last couple of backbone layers

- Stronger overfitting observed
- No conclusive results, needs more hyperparam tuning
- TODO: Run together with LC dropout

Implemented support for pretraining LST on masked LM tasks

- Haven't had time to run it
- Would take >1 day for a minimum number of steps
- Pretraining on the whole corpus unfeasible on available hardware
- Original RoBERTa: 1024 (!! ) V100s for 1 day
  - 500k steps @ 8k batch size
  - => 250 mil steps at 16 batch size (what we mostly use)
  - We could maybe do 0.1% of that...

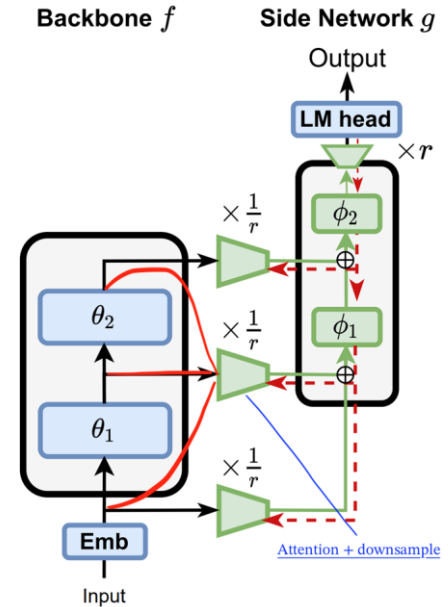


# k-LST + Ladder connection dropout

Implemented ladder connection dropout

- During training draw k-length mask of probability  $p$ , mask the connection(s)
- Observed less overfitting => Assumption was correct
- Slightly better results than without dropout

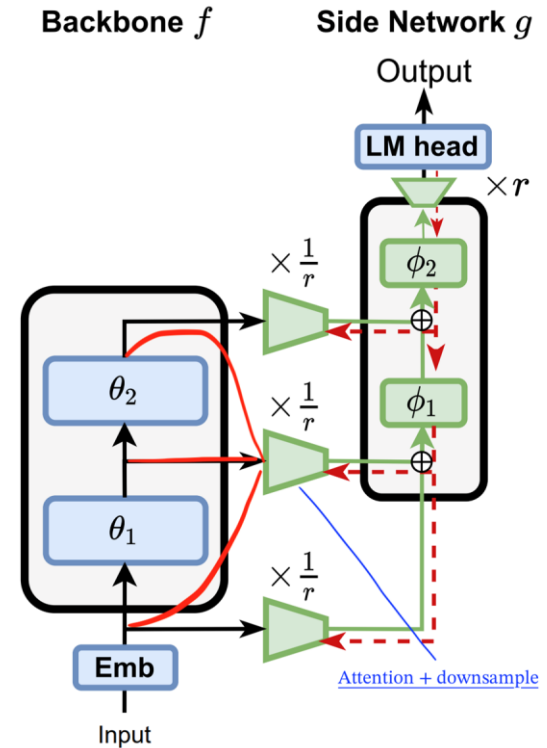
Method	SST-2 Accuracy	SST-2 F1
RoBERTa-large + LST	93.12%	93.10%
RoBERTa-large + 9-LST	94.50%	94.62%
RoBERTa-large + 9-LST + Dropout	94.84%	95.00%
RoBERTa-large + Full FT	96.40%	-



# k-LST - What's left?

What's left?

- More hyperparameter tuning
- Smarter weight initialization
- Pretrain on Masked Language Modelling prior to FT
- Start fixing seeds for replicable final results
- Diagram for poster and report
- From last presentation: Try different downsampling strategies
- Maybe more ideas if they come up...



# MeZO + LST

MeZO + LST not better than standard LST

New Idea: **Modify the inputs.**

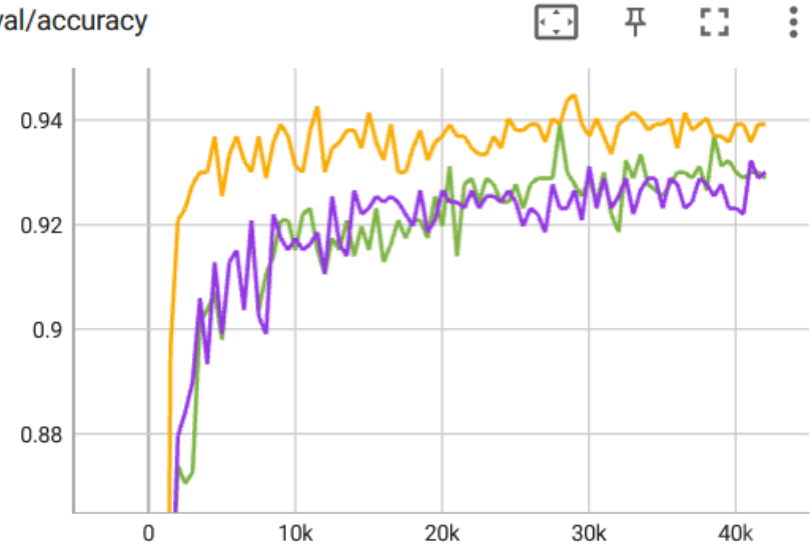
- Make them similar to MeZO fine-tuning by adding the template part.

[CLS] No reason to watch. *It was* [MASK] . [SEP]

Input Template

- MeZO + LST now improves over standard LST
- Problem: Slight improvement for much effort:  
94.5% vs 93.92% max.
- Even smaller difference on LST configurations that perform better (e.g. 9-LST w. dropout)

eval/accuracy





# MeZO + Weight Update Skipping

Idea:

- First fine-tune using MeZO
- Then fine-tune only the last few layers with backpropagation

MeZO seems to be improving

Results similar to LST:

Tradeoff memory-accuracy

	SST-2 Accuracy / F1	Peak Memory (MiB)
Last 1 Layer	92.66 / 92.69	1634
MeZO + Last 1 Layer	93.81 / 93.91	1804
Last 3 Layers	94.15 / 94.33	2122
MeZO + Last 3 Layers	94.84 / 94.86	2122
MeZO + LST	94.50 / 94.58	2044
MeZO + 9-LST	94.95 / 95.09	2428

# MeZO Next Steps

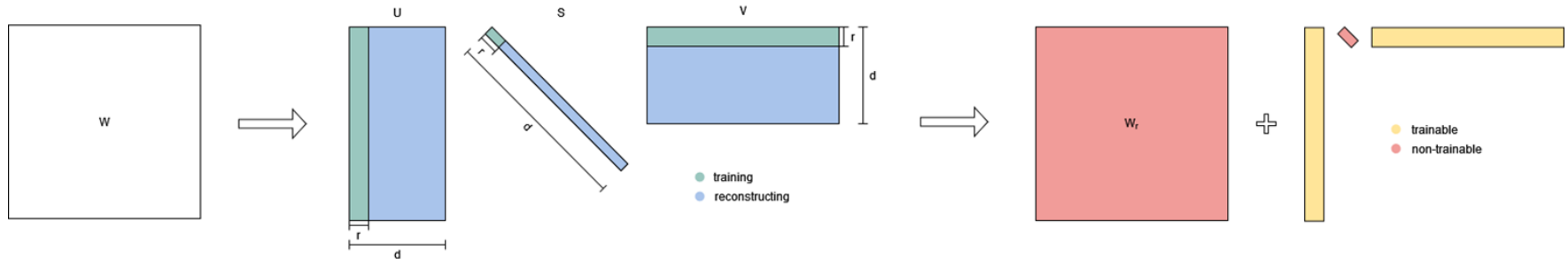
We have trained MeZO for 24000 steps and used that checkpoint for all of our experiments.

It would be interesting to know:

- How much more can MeZO improve on its own
  - For now, it gets about 90.5% accuracy
  - Would be nice to have for comparison with MeZO + LST and MeZO + WUS
  - Also can potentially further improve those methods
- Can earlier MeZO checkpoints provide a similar improvement to LST/WUS
  - Could greatly reduce the time requirements for those methods

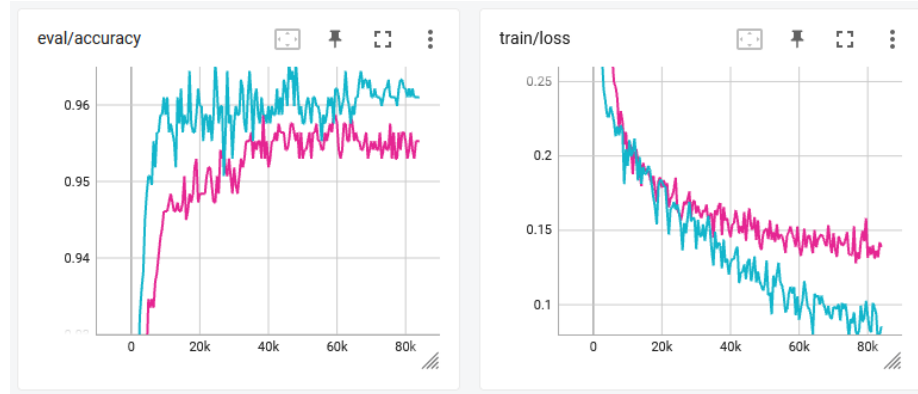
# Singular Value Adaptation (SiVA)

- Another method to update weight matrices in a lower rank
  - Decompose the original weight matrix into  $U$ ,  $S$ ,  $V$  matrices using SVD
  - Split  $U$ ,  $S$ ,  $V$  into smaller training (using the highest  $r$  singular values) and reconstruction (rank  $d$ ) matrices
  - Reconstruct the original model without using the trainable part
  - Train the lower rank  $U$  and  $V$  matrices
  - During forward pass, reconstruct trainable  $U$ ,  $S$ ,  $V$  and add it to the original matrix
- Since there is no random initialization. it should be faster than LoRA



# SiVA Results

- Much faster convergence than LoRA
- Slightly faster training times (5%, 62.63 samples per second vs 59.6)
- As good accuracy/f1 as full fine-tuning, (96.56)



— SiVA  
— LoRA

# SiVA Results

Method	Accuracy	F1 Score	Memory	Training time
Full FT	96.44%	96.53%	8195 MB	3.77 hrs*
LoRA	96.22%	96.29%	4526 MB	2.97 hrs
SiVA	96.56%	96.65%	4529 MB	2.83 hrs
SiVA-kv**	95.76%	95.86%	2920 MB	2.32 hrs

\* estimate based on a different device and lower number of epochs

\*\*only train key and value layers in self attention for lower memory consumption and faster training

# Open Issues & Solutions

- Issues
  - The structure of the poster gets too complex
  - Can't compare our methods with each other
- Solutions
  - Split the methods section into 4: introduction, mezo like models, k-lst, and SiVA
  - Fine-tune on Colab with the same GPU and number of epochs
    - Need to use less than 10 epochs to be able to fully fine-tune Roberta large without getting kicked out

# Plan for the Next Two Weeks

- Finalize experiments & results
  - Train on the same hardware to get speed comparisons
- Finish the poster
- Work on the report