

Memory Efficient PEFT with k-Ladder Side Tuning

Mato Gudelj✉, Coşku Barış Coşlu✉, and Baykam Say✉

School for Computation, Information and Technology, Technical University of Munich

✉ mato.gudelj@tum.de

✉ baris.coslu@tum.de

✉ baykam.say@tum.de

August 6, 2023

Abstract — This work presents k-Ladder Side-Tuning (k-LST), a novel PEFT method that significantly improves upon current state-of-the-art in memory-efficient fine tuning. k-LST closes the performance gap of memory-efficient PEFT methods while retaining a low memory footprint. It employs a sliding window approach to extract backbone features, which are queried by a side network using cross-attention. Unlike other PEFT approaches, k-LST does not directly affect the forward pass of the base model, as it acts as a side network on intermediate features using lateral connections. This allows for efficient hosting in large scale deployments. We show close to full fine-tuning performance at 28% of the memory footprint.

affecting the backbone’s features, limiting its ability to fully leverage the pre-training of the base model.

In this work, we introduce *k*-Ladder Side-Tuning (*k*-LST), a generalization of LST that aims to close the performance gap while maintaining a low memory footprint. *k*-LST extends the original approach by taking a sliding window of *k* backbone features and employing a more intricate fusion approach through cross-attention. We demonstrate that *k*-LST significantly improves performance compared to LST, bringing it closer to other PEFT methods.

Additionally, we explore the impact of pre-fine-tuning with MeZO [4], a low-memory optimizer, and modifying model prompts to produce more informative intermediate features for use by the side network.

1 Introduction

Transformer-based language models have revolutionized the field of natural language processing (NLP), demonstrating remarkable performance across a wide range of tasks. However, the computational cost associated with fine-tuning these large-scale models for specialized tasks can be prohibitive. Parameter-Efficient Fine-Tuning (PEFT) has emerged as a promising solution to this challenge, enabling the adaptation of pre-trained models to new tasks without the need for full fine-tuning.

Among PEFT techniques, additive methods have shown significant promise in terms of both efficiency and performance. In particular, Ladder Side-Tuning (LST) [7], a state-of-the-art additive method, stands out for its memory efficiency. LST involves training a small, separate ladder side network that takes intermediate activations from the backbone network via lateral connections, circumventing the need to backpropagate through the large pre-trained backbone.

While LST represents a significant advancement in memory-efficient PEFT, its performance still lags behind other state-of-the-art methods. We hypothesize that this performance gap is due to the architectural design of LST, which prevents the method from directly

2 Related Work

2.1 LST

Ladder Side-Tuning (LST) [7] is a recent advancement in the area of memory-efficient PEFT that has a significantly smaller memory footprint compared to other methods. The method achieves memory efficiency by circumventing the need to backpropagate through the large pre-trained backbone. Unlike other techniques that incorporate parameters into the pre-trained backbone network, LST involves training a small, separate ladder side network. This side network takes intermediate activations from the backbone network via lateral side connections (Shown in Figure 1).

While LST represents a significant advancement in terms of memory efficiency during training, it achieves significantly lower performance compared to other PEFT approaches. We hypothesize this comes from the architectural design of LST, which prevents the network from directly affecting the backbone’s forward pass. This appears to limit its ability to fully leverage the full pre-training of the original model. Closing this gap would allow the use of memory-efficient PEFT methods in memory-constrained scenarios without significant loss in performance.

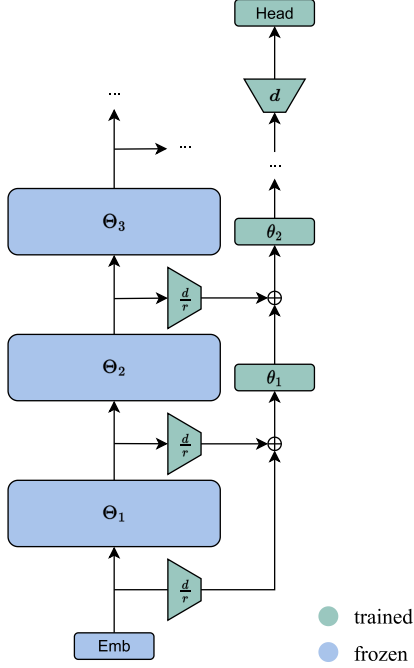


Figure 1 The original LST architecture for an encoder-only backbone. The intermediate backbone representations are downsampled to a dimension of d/r , after which they are fused with the side network features. The gradients of the trainable parameters (shown in green) are computed without going through the backbone network (shown in blue).

2.2 MeZO

Zeroth-order optimization (ZO-SGD [5]) estimates gradients using only forward-passes, making it possible to update neural networks without backpropagation. Memory-efficient Zeroth-order Optimizer (MeZO [4]) is an adaptation of the classical zeroth-order optimization method, reducing its memory consumption by operating in-place. This results in a method that is able to fine-tune LMs with the same memory footprint as inference.

To estimate gradients, MeZO uses Simultaneous Perturbation Stochastic Approximation (SPSA), which requires two forward passes to estimate the gradient. For each forward pass, the parameters of the model are perturbed by different amounts, depending on a random number sampled for each parameter and the perturbation scale, which is a hyperparameter of the model. For loss function $\mathcal{L}$, model parameters $\theta \in \mathbb{R}^d$ and minibatch $\mathcal{B}$, SPSA estimates gradient as:

$$\hat{\nabla} \mathcal{L}(\theta; \mathcal{B}) = \frac{\mathcal{L}(\theta + \epsilon \mathbf{z}; \mathcal{B}) - \mathcal{L}(\theta - \epsilon \mathbf{z}; \mathcal{B})}{2\epsilon} \mathbf{z} \quad (1)$$

where ϵ is the perturbation scale and $\mathbf{z} \in \mathbb{R}^d$ is randomly sampled with $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_d)$.

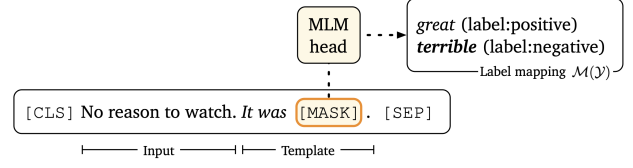


Figure 2 Prompt-based fine-tuning on SST-2. Adapted from [1]. The prompt consists of the input sentence and the template "It was [MASK].". The model is trained to predict the label word "great" for positive input sentences and "terrible" for negative input sentences.

Using the SPSA estimate for gradients, the model is updated with SGD:

$$\theta_{t+1} = \theta_t - \eta \hat{\nabla} \mathcal{L}(\theta; \mathcal{B}_t) \quad (2)$$

where η is the learning rate.

MeZO's in-place implementation saves additional memory for this approach. For each batch, MeZO samples a random seed and every time a parameter gets updated or perturbed for the gradient estimation, it samples the random value $\mathbf{z}$ using this seed. This way, $\mathbf{z} \in \mathbb{R}^d$ does not need to be stored, theoretically reducing memory consumption to the same as inference. In practice, entire weight matrices are perturbed instead of individual scalars, saving time but costing memory equivalent to the largest weight matrix.

2.2.1 Prompt-based Fine-tuning

For each example in the dataset, a model is given a prompt as input; this prompt is a template attached to the input sentence and varies depending on the task and dataset. The template contains a mask token and the model is trained to predict the masked word from a set of label words, where each label word is assigned to one label. Classification is then performed using the logits of the label words on the mask token. Figure 2 shows how prompt-based fine-tuning works on the SST-2 dataset.

3 Methods

This section is split into two subsections. The first describes k-Ladder Side Tuning, while the second explores the application of MeZO and prefix prompts in further improving LST performance.

3.1 k-Ladder Side-Tuning (k-LST)

In an attempt to close the performance gap for memory efficient PEFT, we introduce k-Ladder Side-Tuning (k-

LST). This is a generalization of Ladder Side-Tuning (LST) [7], where LST can be considered a special case of our generalized method for $k = 1$. Each block of the LST side network takes the corresponding intermediate activation from the backbone as a side input. The k-LST extends this approach by taking a sliding window of k backbone features, as well as employing a more intricate fusion approach through cross attention. We demonstrate that the proposed method significantly closes the performance gap to other PEFT methods while retaining a low memory footprint.

3.1.1 Sliding k-window

The k-LST method employs backbone features extracted from a k sized sliding window (see Figure 3). The window for the i -th side feature is centered at the i -th backbone block. For cases when the window overflows we employ padding with zeros. As in [7], we downsample each of these features to dimension $\frac{d}{r}$, where r is the reduction factor hyperparameter. This is done with a learned linear projection $C_i = d_j(B_j)$, where C_i is the i -th feature in the window. The weights of the linear projections are *not* shared.

3.1.2 Feature Fusion With Cross Attention

These downsampled features are queried using a cross attention mechanism. During training, we use random dropout [6] with a probability of p and add a trainable shared positional encoding $\vec{p}_i$ to encode the location of each feature within the k -window:

$$\tilde{C}_i = \text{Dropout}(C_i, \text{is_training}) + \vec{p}_i. \quad (3)$$

To construct the final matrix, we concatenate the resulting features $\tilde{C}_i$ without adding a new axis:

$$C_{j-w:j+w} = \begin{bmatrix} \tilde{C}_1 \\ \tilde{C}_2 \\ \vdots \\ \tilde{C}_k \end{bmatrix}, \quad (4)$$

where j is the index of the center and $w = \lfloor \frac{k}{2} \rfloor$.

For the cross attention operation, we derive the query vectors from the $j - 1$ -th layer of the side network, symbolized as θ_{j-1} . The key and value vectors are obtained from the concatenated features $C_{j-w:j+w}$:

$$Q_j = \theta_{j-1} \quad (5)$$

$$K_j = V_j = C_{j-w:j+w}. \quad (6)$$

Finally, the output of the j -th layer of the side network θ_j is computed using the cross attention mechanism, followed by a side block:

$$X_j = \text{CrossAttention}(Q_j W^q, K_j W^k, V_j W^v) \quad (7)$$

$$\theta_j = \text{SideBlock}_j(X_j) \quad (8)$$

The side block is a simple transformer encoder block, mirroring the architecture of the backbone. If θ_j is the output of the last block before the output, we up-sample the feature with a linear up-projection and pass it to the classification head (Illustrated in Figure 3).

3.1.3 Optimal Hyperparameters

k-LST has three main hyperparameters: the window size k , the reduction factor r and the feature dropout probability p . These three parameters provide granular control over the trade-off between memory footprint and performance. A window size of $k = 9$ showed to be the best performing in our tests. We observed that a higher k is more susceptible to overfitting. Therefore, for the $k = 9$ case, we use a feature dropout probability of $p = 0.2$, while for other experiments we use no dropout. All experiments use a reduction factor $r = 8$. 9-LST is trained with linear LR decay with warmup. The peak LR is $\eta = 0.0002$ with a linear decay to $2e-6$ over 5 epochs and a warmup ratio of 0.12. All experiments use the AdamW optimizer [3] and the RoBERTa-large [2] backbone.

3.1.4 Benefits for Large Scale Deployments

The backbone model is completely independent of the side network. This is in contrast to other widely used PEFT methods where the forward pass of the backbone network is directly influenced by the added PEFT parameters. This is a desirable characteristic for large-scale deployment when many different fine-tuned versions of the same backbone model are being served simultaneously. The benefit is derived from the fact that the forward pass of the large backbone model remains constant, regardless of the fine-tuned version. This allows for the batching of requests irrespective of the specific fine-tuned versions. It's a one-way relationship, where the side network has no effect on the output of the backbone model, it simply consumes the intermediate features to produce the final result. As a consequence, there is no need to clone (or dynamically alter) the backbone model and instead all requests can pass through the same static backbone model, before the intermediate features are fed into the smaller side networks which can run on smaller compute nodes.

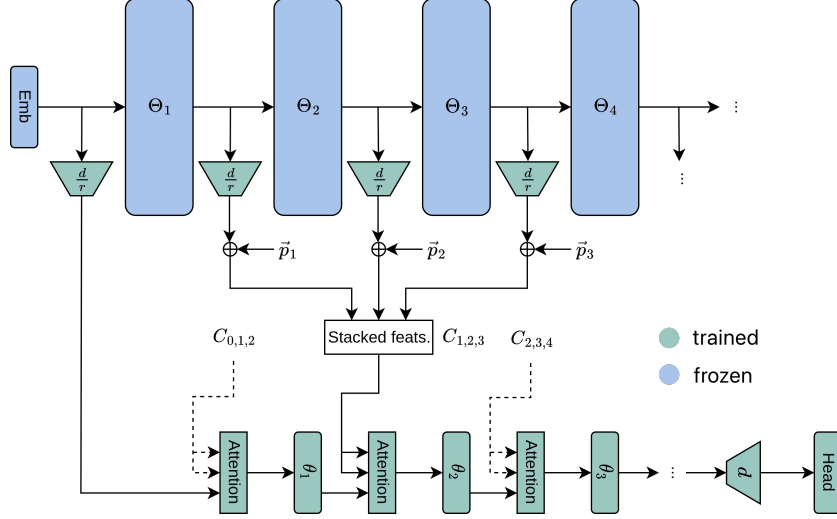


Figure 3 This figure shows the k-LST diagram for $k = 3$. Only the added parameters and head are trained (shown in green), while the backbone remains frozen (shown in blue). We downsample each feature with a linear projection, before adding the positional encoding and concatenating the features into a single matrix. The features are queried with cross attention by the side network. The backbone’s forward pass is completely independent from the side network. The gradients of the trainable parameters can be computed without going through the backbone, resulting in large memory savings.

3.2 MeZO and Training with Prompts

While LST methods provide a significant reduction in memory usage, they also suffer from a reduced accuracy, which is their main weakness. The same weakness applies to layer freezing: While freezing most of the model and training only the last few layers is a very simple yet effective PEFT method, a considerable sacrifice in performance is required in order to save memory on the same level as LST.

Table 1 Experiment results using MeZO and input prompts. Adding an input prompt greatly improves performance for all methods.

Method	Accuracy
MeZO	91.74%
LST	93.12%
LST + MeZO	93.46%
LST + Prompt	94.84%
LST + MeZO + Prompt	94.38%
9-LST	95.82%
9-LST + MeZO	95.18%
9-LST + Prompt	96.01%
9-LST + MeZO + Prompt	94.72%

One of the focal points of our work is trying to bring the performance of these methods closer to full fine-tuning without adding additional memory consumption. Thus, we experiment with the idea of incorporating MeZO into the training process.

Our hypothesis is that the performance of LST can be improved by fine-tuning the backbone network using MeZO before fine-tuning the side network. Since MeZO is very memory-efficient, this approach would not increase memory consumption, while presumably improving performance through better features from the backbone model.

We fine-tune a RoBERTa-large model on the SST-2 dataset using the recipe described in [4]: With $\langle S \rangle$ being the sentence from the SST-2 dataset, We use the prompt " $\langle S \rangle$ It was [MASK]." and our label words are "great" and "terrible". Under this prompt-based fine-tuning setting, we fine-tune using the MeZO algorithm for 48K steps with a constant learning rate of $\eta = 1e-6$, a perturbation scale of $\epsilon = 1e-3$ and a batch size of 16. We then save the fine-tuned model and use the saved model as the backbone to fine-tune a side network. We fine-tune LST and 9-LST models using the configuration described in Section 3.1.3. When fine-tuning the side network we also try giving the prompt that was used when fine-tuning MeZO as input to the model. That means we try both " $\langle S \rangle$ " and " $\langle S \rangle$ It was [MASK]." as inputs. Our intermediate results are shown in Table 1.

For layer freezing, we train the classifier head and the last 3 transformer layers of the model using an AdamW optimizer with a learning rate of $2e-5$, a linear learning rate schedule with a warmup ratio of 0.06 and a weight decay of 0.01.

We compare the results with and without prior MeZO fine-tuning and find that it is uncertain if it

Table 2 Comparison of different methods when fine-tuning a RoBERTa-large model on the SST-2 dataset using a batch size of 16. Samples per Second is measured on an RTX 3070 Mobile and computed as an average over the training process.

Method	Accuracy	F1	Memory Usage (MB)	Samples per Second
Full Fine-Tuning	96.44	96.52	8188	51.8
LoRA	96.22	96.30	4526	64.6
LST	93.23	93.33	1779	73.8
Last 3 Layers	94.04	94.18	2117	179.6
5-LST	94.04	94.04	2209	60.3
9-LST	95.82	95.98	2381	52.4
LST + Prompt	94.84	94.98	1803	73.8
9-LST + Prompt	96.01	96.04	2428	52.4
Last 3 Layers + Prompt	94.84	94.87	2122	179.6

has a positive impact on model performance. Input prompts are the main contributing factor to improved performance, which we hypothesize is due to the priming resulting in more informative backbone features.

4 Results

Table 2 shows a comparison of our methods and their respective baselines. We evaluate the performance in terms of accuracy and F1-score on the SST-2 dataset, as well as peak memory usage and training samples per second. For all experiments, we fine-tune a RoBERTa-large model using a batch size of 16. k-LST stands out for its ability to significantly reduce memory usage while incurring minimal loss in model performance.

5 Conclusion and Future Work

We introduced k-LST as a low-memory PEFT method that extends the original Lateral Side Training (LST) framework. By employing a k-sized window for extracting backbone features and using cross-attention in the side network, k-LST outperforms the original LST in both speed of convergence and achieved accuracy, all the while maintaining a minimal memory footprint. When memory efficiency is a priority, we recommend k-LST coupled with modified prompts for fine-tuning.

The general architecture of k-LST presents a rich avenue for future work. Experimentation with techniques like weight sharing and LayerDrop, akin to those in the original LST, could further reduce computational and memory costs. Pretraining the side network jointly with the backbone can be another promising research direction, potentially enabling the side

network to better utilize backbone features in a general context before specific fine-tuning.

References

- [1] Tianyu Gao, Adam Fisch, and Danqi Chen. *Making Pre-trained Language Models Better Few-shot Learners*. 2021. arXiv: 2012.15723 [cs.CL].
- [2] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: <http://arxiv.org/abs/1907.11692>.
- [3] Ilya Loshchilov and Frank Hutter. “Fixing Weight Decay Regularization in Adam”. In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: <http://arxiv.org/abs/1711.05101>.
- [4] Sadhika Malladi et al. *Fine-Tuning Language Models with Just Forward Passes*. 2023. arXiv: 2305.17333 [cs.LG].
- [5] J.C. Spall. “Multivariate stochastic approximation using a simultaneous perturbation gradient approximation”. In: *IEEE Transactions on Automatic Control* 37.3 (1992), pp. 332–341. DOI: 10.1109/9.119632.
- [6] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [7] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. *LST: Ladder Side-Tuning for Parameter and Memory Efficient Transfer Learning*. 2022. arXiv: 2206.06522 [cs.CL].