Generalizable and Stable Finetuning of Pretrained Language Models on Low-Resource Texts

Sai Ashish Somayajula Youwei Liang Li Zhang Abhishek Singh Pengtao Xie

University of California, San Diego

Introduction

Pretrained language models (PLMs) have significantly improved the performance on various NLP tasks

Finetuning PLMs on low-resource texts poses significant challenges

- High variance in performance for different final layer weight initializations
- Prone to overfitting leading to poor generalization on test set

Prior Methods

Encourage proximity to pretrained weights

- Weight Decay [1], RecAdam [2], Top-K-layer Finetuning [3], Mixout [4]
- Finetuning only a sub-network chosen based on *empirical* Fisher Information matrix (FIM)
  - Child-Tuning$_D$ [5], DPS Dense [6]
  - Promising direction with improved results

Prior Methods

Empirical FIM-based sub-network selection: Green edges indicate weights that are finetuned on the downstream task, while red edges indicate frozen pretrained weights.

Limitations

Low-resource scenarios

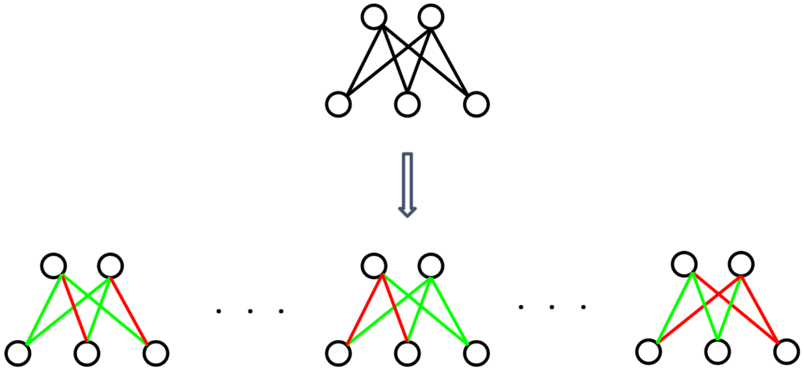
- Data scarcity can skew the gradients used to compute the FIM, leading to sub-optimal sub-network selection [7].
- [8] theoretically shows that empirically determined FIM deviates significantly from the true FIM when sample size is low.

Method: Motivation

Deviate from prior empirical FIM based selection.

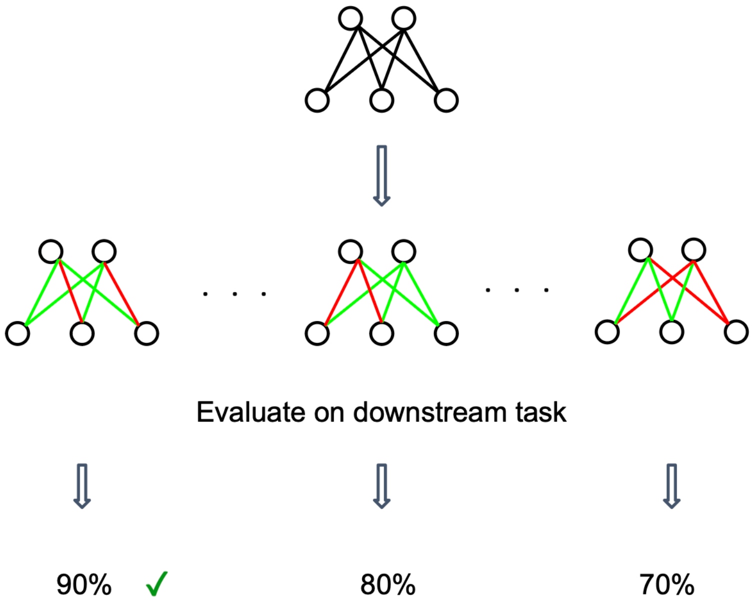
Method: Motivation

Deviate from prior empirical FIM based selection.



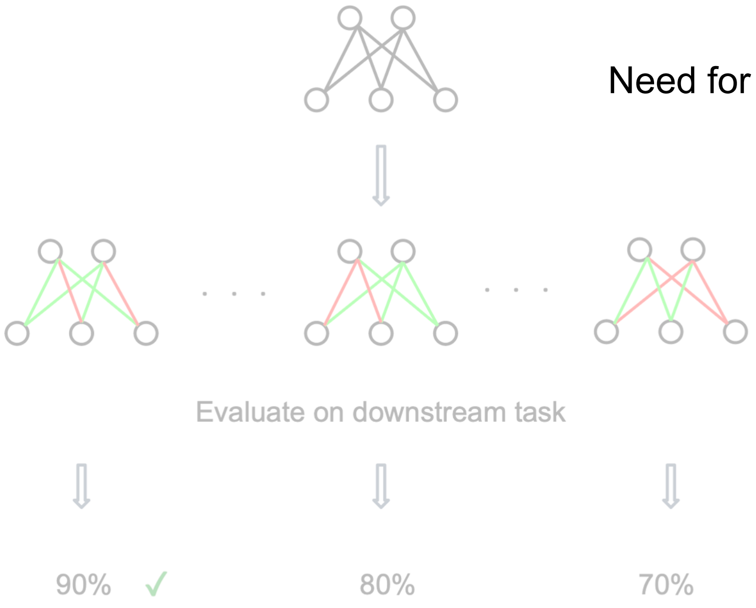
Many choices for sub-networks! How to select an optimal one?

Method: Motivation



A combinatorial problem of selecting an optimal sub-network based on the performance on downstream task.

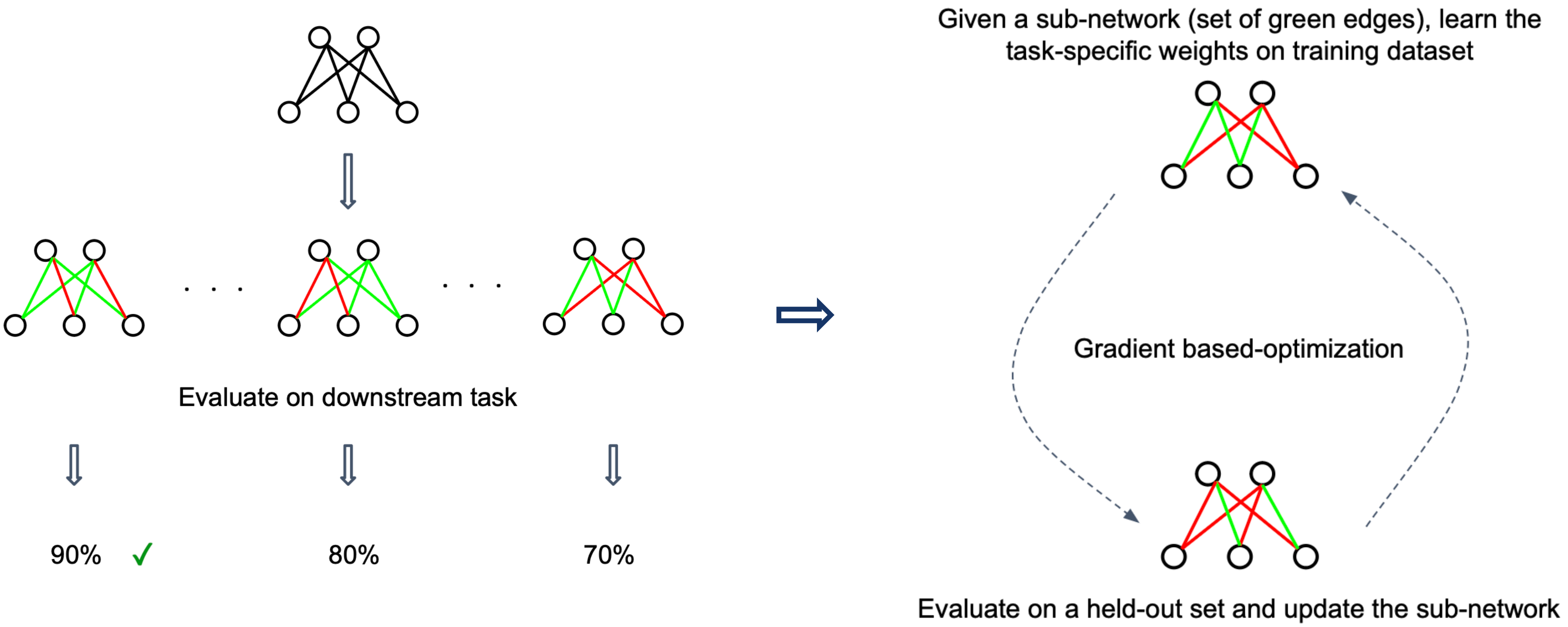
Method: Motivation



Need for computationally efficient approach!

A combinatorial problem of selecting an optimal sub-network based on the performance on downstream task.

Method: Motivation



Continuous optimization of sub-network and task-specific weights based on the downstream task performance

Method: Attention Guided Weights Mixup

For a given sub-network, W_0 are the pretrained weights and W are the task-specific weights

Weights on the red edges can be written as, $\mathbf{0} \odot W + \mathbf{1} \odot W_0$

Weights on the green edges can be written as, $\mathbf{1} \odot W + \mathbf{0} \odot W_0$

$\odot$ denotes the element-wise multiplication operation

Method: Attention Guided Weights Mixup

$$\widehat{W} = g(W, \alpha, W_0) = \alpha \odot W + (1 - \alpha) W_0$$

- Represent each weight as a linear interpolation of pretrained weight W_0 and task-specific weight W
- α referred to '*attention parameters*' in this work, determines the chosen sub-network
  - If α = **1** then the corresponding weight belongs to the sub-network to be finetuned
  - If α = **0** then the corresponding weight is assigned to the frozen pretrained weight
- W depends on the chosen sub-network, i.e., α
- In this formulation, $\alpha \in [0,1]$ allowing a transition from discrete to continuous sub-network selection

Method: Attention Guided Weights Mixup

$\alpha \in [0,1]$

- $\alpha \in [0,1]$
  - Continuous relaxation of sub-network selection
  - Edges are depicted in shades of red and green
  - If α closer to 0 then the corresponding weight has more influence of pre-trained weight and vice-versa
- Goal is to learn α that influences the sub-network and task-specific weights W

Method: Bi-Level Optimization

Learn α and W, which are interdependent, to improve downstream task performance

- Given a sub-network determined by α, learn W
- Evaluate learned network determined by W, update α

Method: Bi-Level Optimization

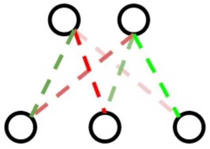
Learn α and W, which are interdependent, to improve downstream task performance

- Given a sub-network determined by α, learn W
- Evaluate learned network determined by W, update α

Bi-level optimization framework!

Method: Bi-Level Optimization

Stage 1 – Given a sub-network determined by α, learn W on the training split



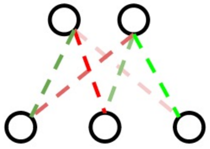
Task weights W
as a function of α

$$\arg\min_{W} \mathcal{L}(g(W, \alpha, W_0); \mathcal{D}^{\mathrm{B-tr}}) + \lambda_1 ||W||_F^2 \Rightarrow W^*(\alpha)$$

Training split $\mathcal{D}^{\mathrm{B-tr}}$

Method: Bi-Level Optimization

Stage 2 – Evaluate learned network determined by $W^*(\alpha)$ on the validation split, update α



Learned task weights $W^*(\alpha)$

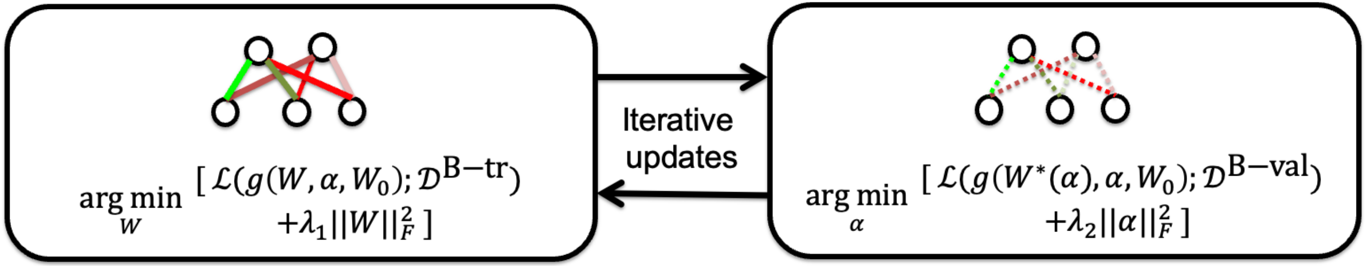
$$\arg\min_{\alpha} \mathcal{L}(g(W^*(\alpha), \alpha, W_0); \mathcal{D}^{\mathrm{B-val}}) + \lambda_2 ||\alpha||_F^2 \Rightarrow \alpha^*$$

Validation split $\mathcal{D}^{\mathrm{B-val}}$

Method: Bi-Level Optimization

$$\min_{\alpha} \quad \mathcal{L}(g(W^*(\alpha), \alpha, W_0); \mathcal{D}^{\mathrm{B-val}}) + \lambda_2 ||\alpha||_F^2$$

$$\text{s.t.} \quad W^*(\alpha) = \arg\min_{W} \mathcal{L}(g(W, \alpha, W_0); \mathcal{D}^{\mathrm{B-tr}}) + \lambda_1 ||W||_F^2$$



Stage 1: Learn W on the train split

Stage 2: Evaluate on validation split and update α

Method: Bi-Level Optimization

Iteratively use one-step gradient descent and finite-difference approximation to solve the optimization [9]

$$W^*(\alpha) \approx W' = W - \eta_w \nabla_{\mathrm{W}}[\mathcal{L}(g(W, \alpha, W_0); \mathcal{D}^{\mathrm{B-tr}}) + \lambda_1 ||W||_F^2]$$

$$\alpha^* \approx \alpha' = \alpha - \eta_\alpha \nabla_\alpha[\mathcal{L}(g(W', \alpha, W_0); \mathcal{D}^{\mathrm{B-val}}) + \lambda_2 ||\alpha||_F^2]$$

Method: Implementation Details

Use low-rank approximation of α, express as product of two rank 1 matrices

Split original training dataset in 4:1 ratio to obtain training $\mathcal{D}^{\mathrm{B-tr}}$ and validation splits $\mathcal{D}^{\mathrm{B-val}}$

Perform random sampling K times and average the learned α and W parameters to mitigate overfitting (K = {1, 2, 5})

Please refer to the paper for more details!

Results

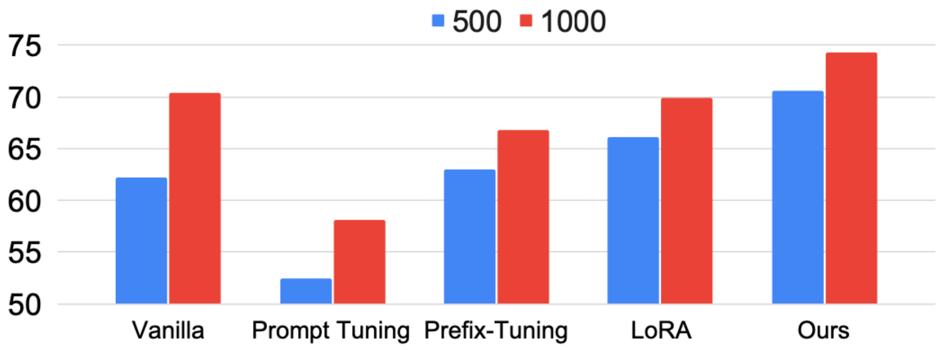
Comparison with FIM-based sub-network selection methods on low-resource scenarios

Training split	**Vanilla**	**CHILD-TUNING$_D$**	**DPS Dense**	**Ours**
300	62.54 ± 6.57	62.47 ± 5.5	61.69 ± 5.62	**68.97** ± 3.09
500	65.85 ± 4.57	68.35 ± 4.36	68.99 ± 2.92	**72.42** ± 2.14
1000	73.19 ± 2.62	74.07 ± 2.75	75.00 ± 1.61	**76.68** ± 1.58

Comparison of Our Method with Vanilla, Child-Tuning$_D$, and DPS Dense Method Using BERT Large Across 300, 500, and 1000 Training Data Splits: Averaged Evaluation Metrics Over Eight GLUE Datasets (Highest Performance in Each Row Indicated in Bold)

Results

Comparison with parameter efficient finetuning (PEFT) methods on low-resource scenarios



Averaged Performance Across CoLA, RTE, STSB, and MRPC Datasets for Vanilla, Prompt Tuning, Prefix-Tuning, LoRA, and Our Method Using BERT Large in Low-Resource Scenarios with 500 and 1000 Training Instances

Results

Evaluation across various PLMs

Models	Methods	CoLA		MRPC		RTE		STSB		Average	
		Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
BERT	Vanilla	64.11	1.33	90.80	1.77	70.69	2.83	89.92	0.61	78.88	1.64
	Ours	**66.07**	1.35	**91.84**	0.37	**73.43**	1.52	**90.34**	0.48	**80.42**(+1.54)	0.93(-0.71)
BART	Vanilla	58.54	1.41	92.03	0.73	81.84	1.41	91.54	0.40	80.99	0.99
	Ours	**60.15**	0.81	**92.33**	0.40	**84.26**	0.54	**92.20**	0.09	**82.23**(+1.24)	0.46(-0.53)
RoBERTa	Vanilla	66.06	2.07	92.25	0.57	<u>78.52</u>	<u>13.01</u>	91.89	0.31	82.18	3.99
	Ours	**66.52**	1.45	**92.58**	0.48	**84.22**	1.44	**92.21**	0.08	**83.88**(+1.70)	0.86(-3.13)
DeBERTa	Vanilla	63.74	1.34	92.31	0.37	85.59	1.58	91.74	0.17	83.34	0.86
	Ours	**65.96**	1.15	**92.32**	0.28	**86.17**	1.47	**91.99**	0.15	**84.11**(+0.77)	0.76(-0.10)
XLNet	Vanilla	<u>40.93</u>	<u>27.28</u>	91.83	0.91	<u>71.17</u>	<u>14.40</u>	91.68	0.19	73.90	10.69
	Ours	**61.66**	1.95	**92.19**	0.38	**83.54**	1.44	**92.12**	0.08	**82.38**(+8.48)	0.96(-9.73)

Comparison of Our Method and Vanilla Finetuning on Five Popular PLMs: Evaluation Over Ten Runs with Different Random Seeds, Reported as Mean and Standard Deviation. Average Score Represents Performance Across Four Datasets. Best Scores Highlighted in Bold, Underlined Values Indicate Degenerate Seeds

Results

Comparison with other prior methods

Methods	CoLA		MRPC		RTE		STSB		Average	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Vanilla	64.11	1.33	90.80	1.77	70.69	2.83	89.92	0.61	78.88	1.64
Mixout	64.42	1.51	91.31	1.08	72.05	1.67	90.39	0.57	79.54	1.21
R3F	64.62	1.38	91.63	0.93	70.75	1.76	89.92	0.61	79.23	1.17
R-Dropout	64.14	1.58	**91.87**	0.78	70.24	2.83	90.25	0.49	79.13	1.42
CHILD-TUNING$_D$	64.85	1.32	91.52	0.81	71.69	1.95	90.42	0.44	79.62	1.13
Re-init	64.24	2.03	91.61	0.80	72.44	1.74	**90.71**	0.14	79.75	1.18
DPS Dense	64.98	1.08	91.50	0.83	73.14	1.97	90.51	0.55	80.03	1.11
DPS Dense (Our run)	64.08	1.50	90.25	2.21	71.92	1.45	90.20	0.47	79.11	1.41
Ours	**66.07**	1.35	91.84	0.37	**73.43**	1.52	90.34	0.48	**80.42**	0.93

Comparison of Our Method with Other Regularization Methods on Four Small Datasets (CoLA, RTE, MRPC, STSB): Mean and Standard Deviation of Ten Random Seeds Reported for Each Method. Bold Indicates Best Performance. Double-Sided T-Tests Show Statistically Significant Improvement ($p < 0.05$) Over Vanilla. Baseline Results from DPS Dense [6]

Conclusions

Deviate from prior FIM-based sub-network selection which is sub-optimal in low-resource scenarios

Attention-guided weights mixup strategy for continuous relaxation of sub-network selection and task weights estimation

Bi-level optimization framework to optimize both W and α on different splits of training data

Outperforms various baselines on low-resource scenarios

Demonstrates improved stability across PLM architectures

Future Works

Potential in continual learning

Our method could help models retain old knowledge while learning new tasks.

References

[1] Daumé III, Hal. "Frustratingly easy domain adaptation." *arXiv preprint arXiv:0907.1815* (2009).

[2] Chen, Sanyuan, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. "Recall and learn: Fine-tuning deep pretrained language models with less forgetting." *arXiv preprint arXiv:2004.12651* (2020).

[3] Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. "Parameter-efficient transfer learning for NLP." In International conference on machine learning, pp. 2790-2799. PMLR, 2019.

[4] Lee, Cheolhyoung, Kyunghyun Cho, and Wanmo Kang. "Mixout: Effective regularization to finetune large-scale pretrained language models." arXiv preprint arXiv:1909.11299 (2019).

[5] Xu, Runxin, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. "Raise a child in large language model: Towards effective and generalizable fine-tuning." arXiv preprint arXiv:2109.05687 (2021).

[6] Zhang, Haojie, Ge Li, Jia Li, Zhongjin Zhang, Yuqi Zhu, and Zhi Jin. "Fine-tuning pre-trained language models effectively by optimizing subnetworks adaptively." Advances in Neural Information Processing Systems 35 (2022): 21442-21454.

References

[7] Kunstner, Frederik, Philipp Hennig, and Lukas Balles. "Limitations of the empirical fisher approximation for natural gradient descent." Advances in neural information processing systems 32 (2019).

[8] Soen, Alexander, and Ke Sun. "On the variance of the Fisher information for deep learning." Advances in Neural Information Processing Systems 34 (2021): 5708-5719.

[9] Choe, Sang Keun, Willie Neiswanger, Pengtao Xie, and Eric Xing. "Betty: An automatic differentiation library for multilevel optimization." arXiv preprint arXiv:2207.02849 (2022).