Know Where You're Going: Meta-Learning for Parameter-Efficient Fine-Tuning

Mozhdeh Gheini, Xuezhe Ma, Jonathan May
Information Sciences Institute
University of Southern California
{gheini, xuezhema, jonmay}@isi.edu

Abstract

A recent family of techniques, dubbed lightweight fine-tuning methods, facilitates parameter-efficient transfer by updating only a small set of additional parameters while keeping the parameters of the original model frozen. While proven to be an effective approach, there are no existing studies on if and how such knowledge of the downstream fine-tuning approach calls for complementary measures after pre-training and before fine-tuning. In this work, we show that taking the ultimate choice of fine-tuning into consideration boosts the performance of parameter-efficient fine-tuning. By relying on optimization-based meta-learning using MAML with certain modifications for our distinct purpose, we *prime* the pre-trained model specifically for parameter-efficient fine-tuning, resulting in gains of up to 4.96 points on cross-lingual NER fine-tuning. Our ablation settings and analyses further reveal that the specific approach we take to meta-learning is crucial for the attained gains.[1]

1 Introduction

The pre-training → fine-tuning paradigm is the dominant practice in natural language processing, owing to state-of-the-art performance on a wide variety of tasks (Qiu et al., 2020). The impressive effectiveness of this approach does not come at a low price. It requires iterative adjustment of anywhere between millions (Devlin et al., 2019) to staggering billions of parameters (Chowdhery et al., 2022). With this many parameters, fine-tuning *all* parameters, as is common, becomes exceedingly computationally expensive: where many models need to be fine-tuned, serving a separate copy of all a model's parameters for each instance is costly in terms of storage.

Recent works on parameter-efficient (PE)[2] fine-tuning address this issue by introducing methods that alternatively rely on only changing a tiny set of extra parameters (Houlsby et al., 2019; Li and Liang, 2021; Hambardzumyan et al., 2021; Lester et al., 2021; Hu et al., 2022; He et al., 2022) or a small fraction of the existing model's parameters (Gheini et al., 2021; Ben Zaken et al., 2022). These methods have been shown to be competitive with full fine-tuning despite modifying only as little as 0.01% of all the parameters (Liu et al., 2022).

With this shift towards lightweight fine-tuning, we ask if the pre-training needs to be complemented in any way as well. Ought we further modify the pre-trained model, knowing that we are going to opt for PE fine-tuning? Specifically, can we extend pre-training in a way that leads to *parameter initializations that better suit PE fine-tuning* than the initializations coming outright from the pre-trained language model (PLM) and used by full fine-tuning?

In this work, we show that, in fact, we can use optimization-based meta-learning to further modify the parameters from a PLM so that they are more beneficial for PE fine-tuning and result in improved performance on the target task after transfer. We term this step, which sits between conventional pre-training and fine-tuning, "*priming*." Specifically, as we describe in §3.2, we tweak the popular meta-learning approach MAML (Finn et al., 2017) for priming and crucially simulate the actual PE fine-tuning procedure in the inner loop of the algorithm. This means that instead of including all the parameters in the inner loop gradient update, we only consider those that will be updated by the PE fine-tuning method. Thus, during the meta-gradient update in the outer loop of the algorithm, this information about the ultimate fine-tuning approach will be incorporated into the pre-trained values.

We choose cross-lingual transfer for named entity recognition (NER) as the testbed to show the effectiveness of priming stage. We show that prim-

[1]Our code is available at https://github.com/MGheini/meta-learning-for-peft.

[2]We use descriptors "parameter-efficient" and "lightweight" interchangeably.

Findings of the Association for Computational Linguistics: ACL 2023, pages 11602–11612
July 9-14, 2023 ©2023 Association for Computational Linguistics

ing a PLM boosts the performance of cross-lingual PE fine-tuning for NER by up to 4.96 F1 points. We provide the details of our lightweight fine-tuning setup in §4. Our ablation study in §5.1 reveals that simulating the fine-tuning procedure is indispensable to the observed improvements: it is not meta-learning in general, but *how* we formulate the meta-learning setup that leads to observed gains.

Our **contributions** are: 1) We propose a meta-learning based mechanism termed "priming" to further update the parameters of a PLM in a way that improves the final PE transfer performance; 2) We show the effectiveness of priming for cross-lingual transfer for NER as an exhibit; 3) We justify and shed more light on the importance of the design elements in the priming algorithm through an ablation analysis.

2 Meta-Learning Background

The meta-learning problem can be viewed as acquiring *meta-parameters* θ using meta-training data $\mathcal{D}_{\text{meta-train}}$ such that θ, when used for *adaptation*, improves performance on a new task with training data $\mathcal{D}_{\text{train}}$ (Finn, 2019). Optimization-based meta-learning algorithms formulate adaptation as an optimization procedure during which task parameters ϕ are obtained by fine-tuning meta-parameters θ:

$$\phi = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_{\text{train}}) \tag{1}$$

where $\mathcal{L}$ is the task-dependent loss function.

Under this model of adaptation, meta-learning becomes a search for meta-parameters θ such that when used as initialization, optimal ϕ may be found via fine-tuning over many tasks. During meta-training, a "task" is modeled as a tuple of a training (*support*) set $\mathcal{D}^{\text{tr}}$ and a testing (*query*) set $\mathcal{D}^{\text{ts}}$. Hence, $\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \cdots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$. Specifically, MAML (Finn et al., 2017), which we take inspiration from, moves towards solution $\theta^\star$ for meta-parameters θ through a bi-level optimization procedure:

$$\theta^\star = \underbrace{\arg\min_\theta \sum_{\substack{(\mathcal{D}_i^{\text{tr}}, \mathcal{D}_i^{\text{ts}}) \\ \in \mathcal{D}_{\text{meta-train}}}} \mathcal{L}(\overbrace{\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}})}^{\textit{inner} \text{ optimization loop}}, \mathcal{D}_i^{\text{ts}})}_{\textit{outer} \text{ optimization loop}} \tag{2}$$

where the inner loop takes gradient steps with respect to θ using the support set of each task to obtain task parameters ϕ_i for each one. The outer loop optimization process then takes meta-gradient steps with respect to θ by evaluating post-inner-update performance on the query set of each task, modifying θ to be a better initialization.

3 Priming for Parameter-Efficient Fine-Tuning through Meta-Learning

3.1 Problem Formulation

Provided with a PLM parameterized by parameters θ_p, and a dataset $\mathcal{D}$ for a target task, conventional fine-tuning practice adds a *task-specific head* parameterized by parameters θ_h (initialized randomly) to the PLM and updates all parameters $\theta_p \cup \theta_h$. To avoid such expensive updates with all parameters, PE fine-tuning designates an additional set of parameters (initialized randomly) as θ_a as the only parameters to be updated along θ_h while keeping θ_p frozen. Note that θ_a is deliberately added in such a way that $|\theta_h| + |\theta_a| \ll |\theta_p|$.

With this alteration, perhaps prior to fine-tuning, θ_p can first be further updated to reach $\theta_p^\star$, which, if transferred specifically under the parameter-efficient setting, results in better performance. We call this extra step between pre-training and fine-tuning and the problem of finding such parameters "priming." As an additional benefit, during priming we can also learn parameters $\theta_a^\star$ to be used instead of random initializations θ_a. Priming does not take away the benefits of PE fine-tuning: ultimately fine-tuning still relies on changing (and hence storing) the same number of parameters that would change without priming ($|\theta_h| + |\theta_a^\star|$); it just starts from more suitable initializations $\theta_p^\star$ and $\theta_a^\star$.

3.2 Priming Algorithm

We model priming as an optimization-based meta-learning problem. However, we refrain from directly applying MAML to it. This is due to the key observation that under PE fine-tuning, the adaptation procedure, as shown in Equation 1, has changed: only a subset of parameters are updated during adaptation. Hence, it should be properly simulated in the inner loop in Equation 2. So during priming, we *only* include θ_a and θ_h in the *inner* loop, mimicking PE fine-tuning and do not include θ_p. θ_p and θ_a then receive the meta-gradients in the *outer* loop and change accordingly.

Algorithm 1 outlines the adaptations used for priming. The inner loop (lines 3-8) simulates exactly how we are going to ultimately fine-tune in a lightweight fashion by only updating θ_a and θ_h. The statement marked as red and without a line

Algorithm 1 Priming for Lightweight Fine-Tuning (PE FT)

Require: model $f_{\theta=\theta_p \cup \theta_h \cup \theta_a}$: pre-trained params θ_p, task head params θ_h, and PE FT params θ_a
Require: $\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \cdots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$
Require: $\mathbb{L} = \{\mathcal{L}_1, ..., \mathcal{L}_t\}$: set of loss functions corresponding to all potential different tasks
Require: α, β: learning rates
Require: S: number of inner gradient steps

1: **while** not converged **do**
2: Sample a batch of tasks $\mathcal{T}$
3: **for all** $\mathcal{T}_i \in \mathcal{T}$ **do**
4: $\theta^i = \theta$
5: **for** $s \leftarrow 1, \ldots, S$ **do**
6: $\theta_a^i = \theta_a^i - \alpha \nabla_{\theta_a^i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta^i}, \mathcal{D}_{\mathcal{T}_i}^{\text{tr}})$; $\theta_h^i = \theta_h^i - \alpha \nabla_{\theta_h^i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta^i}, \mathcal{D}_{\mathcal{T}_i}^{\text{tr}})$
$\theta_p^i = \theta_p^i - \alpha \nabla_{\theta_p^i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta^i}, \mathcal{D}_{\mathcal{T}_i}^{\text{tr}})$ ▷ In MAML, but not here as we are simulating PE FT.
7: **end for**
8: **end for**
9: Meta-gradient steps $\theta_a = \theta_a - \beta \nabla_{\theta_a} \Sigma_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta^i}, \mathcal{D}_{\mathcal{T}_i}^{\text{ts}})$;
$\theta_p = \theta_p - \beta \nabla_{\theta_p} \Sigma_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta^i}, \mathcal{D}_{\mathcal{T}_i}^{\text{ts}})$
10: $\theta_h = \theta_h^1$
11: **end while**
12: **return** θ_p, θ_a

number, which additionally updates pre-trained parameters θ_p, would be executed by MAML. But we crucially omit it in our proposed priming algorithm. At the end of the outer loop (line 9), we take meta-gradient steps with respect to the parameters the initializations of which we are trying to enhance, θ_a and θ_p. As θ_h will be initialized from scratch for each new task at the time of fine-tuning, we do not compute meta-gradients for it, and simply assign it to one of the calculated sets in the inner loop, e.g., the first set corresponding to the first task in the sampled batch of tasks ($\theta_h = \theta_h^1$ on line 10).

4 Experimental Setup

While our proposed priming algorithm is model-agnostic, we need a concrete PE fine-tuning and meta-training setup for empirical evaluation.

For lightweight fine-tuning, we choose adapters (Houlsby et al., 2019). In our experiments, we add a single adapter after the last layer of the pre-trained Transformer. Our model then computes the logits for input as: $h(g(f(x; \theta_p); \theta_a); \theta_h)$, where f is the pre-trained model, g is the single adapter layer at the top, and h is the task-specific head.

As a testbed, we experiment with cross-lingual NER. For this case, we can design the priming (meta-learning) and fine-tuning stages as such:
Meta-Learning: Using one or more source languages, we construct the meta dataset and run priming. Per our problem formulation, θ_p and θ_a are shared among languages, but each source language l has a separate head, parameterized by θ_{h_l}.
Fine-Tuning: For each desired target language, we use the pre-trained and adapter parameter initializations acquired during meta-learning along with randomly initialized new head parameters as the model's starting point. We then fine-tune only the adapter parameters and the head parameters. In our single adapter layer setup, this means only updating fewer than **0.4%** of all the parameters.

4.1 Data Details

We use the WikiAnn multilingual NER dataset (Pan et al., 2017), which is available from the Datasets Python library (Lhoest et al., 2021). The train, validation, and test splits, as provided by Rahimi et al. (2019), range from 100 to 20k instances. In our experiments, we use the English and Spanish sets as source languages, each with 20k instances during the priming stage. At fine-tuning, we evaluate the quality of transfer for six target languages: Hindi (5k instances), Afrikaans (5k), Azerbaijani (10k), Lithuanian (10k), Estonian (15k), and Dutch (20k).

		Hindi	Afrikaans	Azerbaijani	Lithuanian	Estonian	Dutch
Without Priming	**1/Full FT** (100%)	**86.73**	**91.29**	**87.70**	**89.43**	**90.88**	**91.47**
	2/HT (3e-3%)	72.71	79.11	74.24	78.34	81.23	78.90
	3/AT (0.4%)	77.76	84.10	81.08	83.00	85.13	83.89
With Priming	**4/Meta Priming → AT**	**81.30**	**87.76**	**82.98**	**86.03**	**86.73**	**88.85**
	5/FT Priming → AT	80.34	87.70	81.74	85.84	86.43	88.61
	6/MP [MAML Loop] → AT	80.15	86.10	81.54	85.66	86.06	88.15
	7/MP [1 Inner Step] → AT	80.54	86.48	80.74	84.87	86.43	88.72

Table 1: Entity-level micro F1 under each of the fine-tuning settings for NER across six languages. Bold numbers indicate top-scoring methods in each category. Percentages next to each setting are the fraction of parameters that are updated (all **AT** settings have the same percentage). Priming as described in this work is most effective in improving PE fine-tuning performance and closing the gap with Full FT. All priming experiments are run twice (including the priming stage), and we report the *average score over two runs*.

4.2 Implementation Details

We use $\text{mBERT}_{\text{BASE}}$ as the PLM. The meta-gradient in the outer loop relies on second-order gradients, which are expensive to compute. Thus, following Finn et al. (2017), we use a first-order approximation in our implementation. For the inner loop, we take five steps of stochastic gradient descent with a learning rate of 0.03. For the outer loop, we use the AdamW optimizer (Loshchilov and Hutter, 2019) with a learning rate of 5e-5 and a linear learning rate scheduler. We provide additional details on implementation and frameworks used in Appendix B.

4.3 Baselines and Method Evaluation Settings

To assess the effectiveness of priming, we run two categories of experiments as listed in Table 1. The setting numbers in the table match those used below (e.g., **1/Full FT** ↔ **1/Full fine-tuning baseline**).

The first category includes no priming:

1/Full fine-tuning baseline corresponds to fine-tuning $\theta_p \cup \theta_h$, where θ_h is initialized randomly. It provides an upper bound for PE fine-tuning, and notably is *not* parameter-efficient.

2/Head tuning (HT) baseline corresponds to freezing θ_p (treating the PLM as a feature extractor) and fine-tuning θ_h, where θ_h is initialized randomly. It provides a lower bound for PE fine-tuning.

3/Adapter tuning (AT) baseline corresponds to fine-tuning $\theta_a \cup \theta_h$. It is the baseline PE fine-tuning, and we investigate if priming improves upon it.

We also experiment with a second category, which incorporates priming:

4/Adapter tuning after priming as proposed corresponds to fine-tuning $\theta_a \cup \theta_h$ where θ_p (frozen) and θ_a are acquired through priming, and θ_h is initialized randomly. Compared to the adapter tuning baseline (3), it measures how much priming can improve PE fine-tuning.

5/Adapter tuning after priming through fine-tuning is the same as setting 4 except that instead of priming as proposed, we simply fine-tune $\theta_p \cup \theta_a \cup \theta_h$ on the same data that would have constructed the meta dataset before proceeding with PE fine-tuning just as in setting 4. This is to illustrate that mere exposure to data during priming is not enough, and treating it as an optimization-based meta-learning problem is beneficial.

Additionally, we have two ablation settings to study the effect of simulating PE fine-tuning in the inner loop and the number of inner steps in priming algorithm, which we will discuss in §5.1 and §5.2.

5 Results and Analysis

Per Table 1, among all PE fine-tuning settings without any priming and those with priming, **4/Meta Priming → AT**, which is the materialization of our priming algorithm, is the best-performing. In comparison with baseline PE fine-tuning (**3/AT**), our approach results in gains of up to 4.96 points, indicating that priming with the knowledge of the ultimate transfer process is substantially helpful. Additionally, the approach results in gains of up to 1.24 points compared to fine-tuning-based priming (**5/FT Priming → AT**), signifying that it is not just a matter of exposure to more data, but a matter of appropriately using the extra exposure to simulate the eventual fine-tuning approach.

5.1 Ablation 1: Substitute MAML Inner Loop

To highlight the importance of the change we introduce in MAML, we run the ablation setting **6/MP**

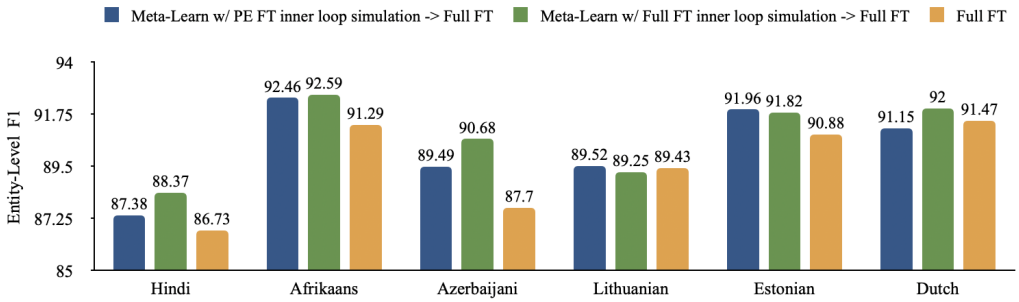

Figure 1: Comparison between different priming strategies for downstream full fine-tuning. In this case, as opposed to parameter-efficient fine-tuning, it is usually beneficial to use full fine-tuning in the inner loop.

[MAML Loop] → **AT** (MP stands for Meta Priming). This is essentially **4/Meta Priming** → **AT** where we update all parameters, and not only those involved in PE fine-tuning, in the inner loop. It can be observed across the board that, in fact, simulating the downstream PE fine-tuning setting is essential for superior performance.

We can also generalize the question at the core of this work: Can we expect gains by using optimization-based meta-learning and simulating the eventual transfer method, whatever it might be?

To determine the answer, we repeat the settings in this section (**4/Meta Priming** → **AT** and **6/MP [MAML Loop]** → **AT**), but replace adapter tuning (**AT**) with full fine-tuning. As shown in Figure 1, in most cases, matching downstream full fine-tuning with a parameter-dense MAML inner loop (green bar in the middle in each series) is superior to mixing it with PE optimization in the inner loop. We hypothesize that the discrepancy in the case of Lithuanian and Estonian is due to the fact that full fine-tuning is powerful, and potentially more robust to heterogeneous priming conditions.

5.2 Ablation 2: Number of Inner Steps

We find that under first-order MAML, the number of inner steps is critical for reaching better initialization. The ablation setting **7/MP [1 Inner Step]** → **AT**, which is identical to **4/Meta Priming** → **AT** with only one inner step, highlights this. **4/Meta Priming** → **AT** with five inner steps, always performs better.

To provide an intuition as to why that is, a visualization of how parameters receive updates under first-order MAML by Wild (2020) is provided in Figure 2. Meta-parameters θ are updated in the direction of the gradient of the query set loss calculated at the value reached at the end of the inner loop. Hence, the fewer the number of inner steps, the more the updates will be similar to those under regular fine-tuning (in the limit of zero inner steps, it will be equivalent to conventional fine-tuning). So additional inner steps are beneficial.

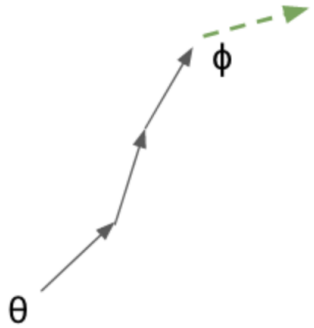

Figure 2: First-order approximation of meta-gradient update. Illustration courtesy of Wild (2020). After three steps of inner updates ($\theta \rightarrow\rightarrow\rightarrow \phi$), θ is updated by the gradient of the query set loss evaluated at ϕ (the green dashed arrow).

6 Conclusion

We propose to add "priming" between the conventional pre-training and parameter-efficient fine-tuning to incorporate awareness of the transfer procedure in the PLM. We model this as optimization-based meta-learning, which integrates such knowledge by updating pre-trained parameters under PE fine-tuning simulation. We show the effectiveness of priming in improving baseline PE fine-tuning on cross-lingual transfer for NER. Further analysis reveals that our decisions to 1) model priming with meta-learning instead of simple fine-tuning and 2) simulate the actual PE fine-tuning in the meta-learning instead of using it unadjusted contribute to the effectiveness of priming.

Limitations

We would like to acknowledge three categories of limitation that we recognize in this work:

- We evaluate the effectiveness of priming in a setting where the tasks used during the priming stage and the fine-tuning stage offer no additional disparity besides being different in language, i.e., they are all NER tasks coming from the same domain. While this degree of variation is consistent with the application of meta-learning in other modalities, e.g., vision (Finn et al., 2017), whether or not the gains we report here remain at the same strength when we introduce diverse tasks during priming and fine-tuning still needs to be tested. Examples of such diversity include strong domain shift or using one task, e.g., POS, for priming and another, e.g., NER, during fine-tuning.

- It's not clear how the size of the pre-trained model affects the necessity of priming. Priming might consistently result in gains, or its benefits might fade away with larger PLMs encoding stronger language capabilities. This also needs to be evaluated.

- Finally, our work does not implement higher-order gradient calculation and does not evaluate and discuss the potential additional gains that might come as a result. That opportunity can be further explored as well.

Acknowledgements

The authors would like to thank their colleagues at CUTELABNAME and USC NLP at large for their support and feedback. The authors also thank anonymous reviewers for their feedback and suggestions, which helped improve this draft. This work is based in part on research sponsored by Air Force Research Laboratory (AFRL) under agreement number FA8750-19-1-1000.

References

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William Falcon and The PyTorch Lightning team. 2019. PyTorch Lightning.

Chelsea Finn. 2019. Meta-learning recipe, black-box adaptation, optimization-based approaches. Last accessed 14 May 2022.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. Cross-attention is all you need: Adapting pretrained

Transformers for machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1765, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: Word-level Adversarial ReProgramming. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4921–4933, Online. Association for Computational Linguistics.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Khurram Javed and Martha White. 2019. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965. Curran Associates, Inc.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. MetaICL: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States. Association for Computational Linguistics.

Farhad Nooralahzadeh, Giannis Bekoulis, Johannes Bjerva, and Isabelle Augenstein. 2020. Zero-shot cross-lingual transfer with meta learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4547–4562, Online. Association for Computational Linguistics.

Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. 2017. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada. Association for Computational Linguistics.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *CoRR*, abs/2003.08271.

Afshin Rahimi, Yuan Li, and Trevor Cohn. 2019. Massively multilingual transfer for NER. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 151–164, Florence, Italy. Association for Computational Linguistics.

Cody Marie Wild. 2020. A search for efficient meta-learning: Mamls, reptiles, and related species. Last accessed 16 May 2022.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu,

Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Mengzhou Xia, Guoqing Zheng, Subhabrata Mukherjee, Milad Shokouhi, Graham Neubig, and Ahmed Hassan Awadallah. 2021. MetaXL: Meta representation transformation for low-resource cross-lingual learning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 499–511, Online. Association for Computational Linguistics.

A Related Work

Our work takes inspiration from and can be contextualized within both the existing lightweight fine-tuning literature and meta-training literature. Lightweight fine-tuning methods are a response to the ever-growing size of the PLMs, which makes full fine-tuning prohibitively expensive. Recently, different flavors of PE fine-tuning have been explored. One category includes methods that add and solely update a new set of parameters; like adapters (Houlsby et al., 2019), prefix tuning (Li and Liang, 2021), and LoRA (Hu et al., 2022), to name a few. Another category of methods does not add any additional parameters and instead relies on updating a small subset of existing parameters of the pre-trained model; for instance, BitFit (Ben Zaken et al., 2022) and exclusive cross-attention fine-tuning (Gheini et al., 2021).

Despite the rich literature on different parameter-efficient transfer approaches, to the best of our knowledge, no existing study investigates whether in response pre-training practices need to be updated in any way. In this work, we attempt to address that void. He et al. (2022) provide a unified framework within which several flavors of lightweight fine-tuning can be interpreted. Therefore we, while studying an adapter-based approach in this work, expect priming to be fundamentally applicable and useful to other flavors too.

We are also inspired by the body of work that takes advantage of optimization-based meta-learning to come by initializations that would be better suited for a specific objective. Xia et al. (2021) use meta-learning to learn representation transformations that transform representations of a high-resource language in a way that they become more beneficial for effective transfer to low-resource languages. Nooralahzadeh et al. (2020) effectively use meta-learning to leverage training data for zero-shot and few-shot cross-lingual transfer on Question Answering and Natural Language Inference. Javed and White (2019) use a meta-objective to optimize representations for continual learning.

Perhaps closest in spirit to our objective and trying to bring these two lines of work together, Min et al. (2022) offer a meta-learning-like solution to "*learn to learn in context*": using our terminology, while we address priming for PE fine-tuning, they address priming for in-context learning (Brown et al., 2020). In-context learning is a few-shot learning technique with no additional training required, where an LM is used to label new instances after conditioning on only a few supervised examples. Min et al. (2022) propose to better prepare the model for such an inference process on a new unseen task by including a tuning stage where the model is trained to do the same on simulated input sequences from a set of available tasks. The extra training stage that they include can be seen as equivalent to our priming stage, where in both cases, the goal is to prepare the model for what is subsequently coming.

B Additional Implementation Details

Our implementation is based off of the Transformers (Wolf et al., 2020) and Lightning (Falcon and The PyTorch Lightning team, 2019) libraries. For our pre-trained model, we use multilingual BERT (mBERT, `bert-base-multilingual-cased`) (Devlin et al., 2019). For the adapter layer, we set the bottleneck dimension as 64 in our experiments.

Our experiments (both priming and fine-tuning stages) are each run on one NVIDIA Quadro RTX 8000 GPU, taking a maximum of twelve hours.

C Licenses of Artifacts Used

We use the following artifacts in compliance with their terms of use:

- WikiAnn dataset by Pan et al. (2017) with splits as provided by Rahimi et al. (2019) under Apache License 2.0
- Transformers (Wolf et al., 2020) under Apache License 2.0
- Lightning (Falcon and The PyTorch Lightning team, 2019) under Apache License 2.0

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section "Limitations" after "Conclusion"

☐ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and Section 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Sections 4.1 and 4.2

☑ B1. Did you cite the creators of artifacts you used?
Sections 4.1 and 4.2 and Appendix C

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Appendix C

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Appendix C

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Section 4.1

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 4.1

C ☑ Did you run computational experiments?

Section 5

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Table 1 and Appendix B

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4.2 and Appendix B

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Table 1

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Not applicable. Left blank.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.