EPFL

Thèse n° 8664

Improving Generalization of Pretrained Language Models

Présentée le 30 mars 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire de systèmes d'information et d'inférence
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Rabeeh KARIMI MAHABADI

Acceptée sur proposition du jury

Prof. E. Bugnion, président du jury
Prof. V. Cevher, Dr J. Henderson, directeurs de thèse
Dr C. Blundell, rapporteur
Prof. L. Zettlemoyer, rapporteur
Prof. A. Bosselut, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

Abstract

Transfer learning, where a language model is first pre-trained on large-scale unlabeled data followed by fine-tuning on a downstream task, has emerged as a dominating technique in natural language processing obtaining the state of the art results on a wide range of tasks. The striking effectiveness of transfer learning has given rise to a variety of methods, and practice. In spite of this rapid progress, transfer learning and building robust models from these pretrained language models (PLMs) which can generalize to unseen domains is a multifaceted problem, requiring addressing several open questions such as a) training models robust to dataset biases, which can generalize better in real-world scenarios b) reducing over-fitting when fine-tuning PLMs on low-resource setting c) fine-tuning strategies allowing learning from multiple training resources while being able to generalize to new domains d) efficient and effective fine-tuning methods for PLMs allowing transfer learning with limited data and to new domains e) few-shot learning with few labeled examples. In this dissertation, we propose multiple methods to improve generalization of PLMs from different aspects:

Our first contribution is to propose two learning strategies to train neural models, which are more robust to dataset biases and transfer better to out-of-domain datasets. We specify the biases in terms of one or more *bias-only models*, which learn to leverage the dataset biases. During training, the bias-only models' predictions are used to adjust the loss of the base model to reduce its reliance on biases by down-weighting the biased examples and focusing training on the *hard* examples. Results show that our debiasing methods greatly improve robustness on several natural language understanding (NLU) benchmarks and improve transfer learning to other textual entailment datasets.

Our second contribution is to propose an effective regularization method to reduce overfitting when fine-tuning PLMs on a small number of training data. We leverage Variational Information Bottleneck (VIB) (Alemi et al., 2017) to suppress irrelevant features when fine-tuning on low-resource target tasks and show that our method effectively reduces overfitting. Moreover, we show that our VIB model finds sentence representations that are more robust to biases in natural language inference datasets, and thereby substantially improves generalization to out-of-domain datasets.

Our third contribution is to develop an effective and parameter-efficient way to fine-tune PLMs in a multi-task learning setup while allowing generalization to new domains. Our method allows sharing information across tasks to enable positive transfer to low-resource and related tasks while avoiding negative task interference. we propose HYPERFORMER++, which employs a compact *hypernetwork* (Ha et al., 2017; Oswald et al., 2020) shared across tasks and layers. The hypernetwork then learns to generate task and layer-specific *adapter* parameters, conditioned on task and layer id embeddings in a transformer model. This parameter-efficient multi-task learning framework allows us to achieve the

best of both worlds by sharing knowledge across tasks via hypernetworks while enabling the model to adapt to each individual task through task-specific adapters. Experiments on the well-known GLUE benchmark show improved performance in multi-task learning while adding only 0.29% parameters per task. We additionally demonstrate substantial performance improvements in few-shot domain generalization across a variety of tasks.

Our fourth contribution is to propose COMPACTER, a method for fine-tuning large-scale language models with a better trade-off between task performance and the number of trainable parameters than prior work. COMPACTER accomplishes this by building on top of ideas from adapters, low-rank optimization (Aghajanyan et al., 2021), and parameterized hypercomplex multiplication layers (Zhang et al., 2021a). Specifically, COMPACTER inserts task-specific weight matrices into a pretrained model's weights, which are computed efficiently as a sum of Kronecker products between shared "slow" weights and "fast" rank-one matrices defined per COMPACTER layer. By only training 0.047% of a pretrained model's parameters, COMPACTER performs on par with standard fine-tuning on GLUE and outperforms standard fine-tuning on SuperGLUE and low-resource settings.

Our final contribution is to propose PERFECT, a simple and efficient method for few-shot fine-tuning of PLMs *without relying on any handcrafting*, which is highly effective given as few as 32 data points. This is in contrast to the current methods for few-shot fine-tuning of pretrained masked language model (PLM) require carefully engineered prompts and verbalizers for each new task, to convert examples into a cloze-format that the PLM can score. PERFECT makes two key design choices: First, we show that manually engineered task prompts can be replaced with *task-specific adapters* that enable sample-efficient fine-tuning and reduce memory and storage costs by roughly factors of 5 and 100, respectively. Second, instead of using handcrafted verbalizers, we learn a new *multi-token label embedding* during fine-tuning which are not tied to the model vocabulary and which allows us to avoid complex auto-regressive decoding. These embeddings are not only learnable from limited data but also enables nearly 100x faster training and inference. Experiments on a wide range of few shot NLP tasks demonstrate that PERFECT, while being simple and efficient, also outperforms existing state-of-the-art few-shot learning methods.

Keywords: transfer learning, generalization, fine-tuning, bias-reduction, multi-task learning, few-shot learning, low-resource setting, parameter-efficient fine-tuning, adapter, robustness.

Abstrakt

Das Transferlernen, bei dem ein Sprachmodell zunächst auf vielen Daten ohne Annotationen vortrainiert und dann auf einer Zielaufgabe feinabgestimmt wird, hat sich zu einer dominierenden Technik in der Verarbeitung natürlicher Sprache entwickelt, und dadurch Ergebnisse auf dem neuesten Stand der Technik für eine Vielzahl von Aufgaben erzielt. Die bemerkenswerte Wirksamkeit des Transferlernens hat zu einer Vielzahl von Methoden und Praktiken geführt. Trotz dieses schnellen Fortschritts ist das Transferlernen und die Konstruktion robuster Modelle aus diesen vortrainierten Sprachmodellen (auf Englisch: Pretrained Language Models, PLMs), welche auf neuartige Anwendungsdomäne verallgemeinert werden können, ein vielschichtiges Problem, das die Beantwortung mehrerer offener Fragen erfordert: a) Modelle trainieren, die robust gegenüber Bias in Datensätzen sind, damit sie besser in realistischen Szenarien funktionieren, b) die Reduktion von Überanpassung, wenn man PLMs auf wenigen Daten feinabstimmt, c) Strategien zur Feinabstimmung, die es erlaubt von mehreren Trainingsressourcen zu lernen, während es gleichzeitig auf neuen Domänen generalisiert, d) effiziente und effektive Feinabstimmung für PLMs, die Transferlernen mit weniger Daten und in neuen Domänen erlaubt, und e) Lernen mit wenigen annotierten Beispielen. In dieser Dissertation schlagen wir mehrere Methoden vor, die die Verallgemeinerung von PLMs unter verschiedenen Aspekten verbessern:

Unser erster Beitrag besteht darin, zwei Lernstrategien vorzuschlagen, um neuronale Modelle zu trainieren, die robuster gegenüber Datensatzverzerrungen sind und besser auf Out-of-Domain-Datensätze übertragen werden können. Wir spezifizieren den Bias in Bezug auf ein oder mehrere *Bias-Only-Modelle*, die lernen, den Dataset-Bias zu nutzen. Während des Trainings werden die Vorhersagen der Nur-Bias-Modelle verwendet, um die Zielfunktion des Basismodells anzupassen, um seine Abhängigkeit vom Bias zu verringern, indem die voreingenommenen Beispiele heruntergewichtet und das Training auf die *schwierigen* Beispiele konzentriert wird. Die Ergebnisse zeigen, dass unsere Debiasing-Methoden die Robustheit bei mehreren NLU-Benchmarks (Natural Language Understanding) erheblich verbessern und zusätzlich das Transferlernen auf anderen Datensätzen des textuellen Schließens verbessern.

Unser zweiter Beitrag besteht darin, eine effektive Regularisierungsmethode vorzuschlagen, um die Überanpassung bei der Feinabstimmung von PLMs auf eine kleine Anzahl von Trainingsdaten zu reduzieren. Wir nutzen das Variational Information Bottleneck (VIB) (Alemi et al., 2017), um irrelevante Merkmale bei der Feinabstimmung von Zielaufgaben mit wenig Ressourcen zu unterdrücken, und zeigen, dass unsere Methode eine Überanpassung effektiv reduziert. Darüber hinaus zeigen wir, dass unser VIB-Modell Satzrepräsentationen findet, die robuster gegenüber Verzerrungen in Inferenzdatensätzen natürlicher Sprache sind, und dadurch die Verallgemeinerung auf Out-of-Domain-Datensätze

erheblich verbessert.

Unser dritter Beitrag ist die Entwicklung einer effektiven und parametereffizienten Methode zur Feinabstimmung von PLMs in einem Multi-Task-Lernaufbau, während gleichzeitig eine Verallgemeinerung auf neue Bereiche ermöglicht wird. Unsere Methode ermöglicht das Teilen von Informationen über Aufgaben hinweg, um einen positiven Transfer zu ressourcenarmen und verwandten Aufgaben zu ermöglichen, während negative Aufgabeninterferenzen vermieden werden. Wir schlagen HYPERFORMER++ vor, das ein kompaktes *Hypernetzwerk* (Ha et al., 2017; Oswald et al., 2020) verwendet, das über Aufgaben und Schichten hinweg geteilt wird. Das Hypernetzwerk lernt dann, aufgaben- und schichtspezifische Adapterparameter zu erzeugen, abhängig von Aufgaben- und Schicht-ID-Einbettungen in ein Transformer-Modell. Dieses parametereffiziente Multitask-Lernframework ermöglicht es uns, das Beste aus beiden Welten zu verbinden, indem wir Wissen über Aufgaben über Hypernetzwerke teilen und gleichzeitig ermöglichen, dass sich das Modell durch aufgabenspezifische Adapter an jede einzelne Aufgabe anpasst. Experimente mit der bekannten GLUE-Benchmark zeigen eine verbesserte Leistung beim Multitasking-Lernen, während nur $0{,}29\%$ Parameter pro Aufgabe hinzugefügt werden. Wir demonstrieren außerdem erhebliche Leistungsverbesserungen bei der Domain-Generalisierung mit wenigen Beispielen für eine Vielzahl von Aufgaben.

Unser vierter Beitrag besteht darin, COMPACTER vorzuschlagen, eine Methode zur Feinabstimmung großer Sprachmodelle mit einem besseren Kompromiss zwischen Aufgabenleistung und der Anzahl trainierbarer Parameter als frühere Arbeiten. COMPACTER erreicht dies, indem es auf Ideen von Adaptern, Niedrig-Rang-Optimierung (Aghajanyan et al., 2021) und parametrisierten hyperkomplexen Multiplikationsschichten (Zhang et al., 2021a) aufbaut. Insbesondere fügt COMPACTER aufgabenspezifische Gewichtsmatrizen in die Gewichte eines vortrainierten Modells ein, die effizient als Summe von Kronecker-Produkten zwischen gemeinsamen "langsamen" Gewichten und "schnellen" Rang-Eins-Matrizen berechnet werden, die pro COMPACTER-Schicht definiert sind. Obwohl nur $0{,}047\%$ der Parameter eines vortrainierten Modells trainiert werden, ist COMPACTER auf Augenhöhe mit der standardmäßigen Feinabstimmung auf GLUE und übertrifft die standardmäßige Feinabstimmung auf SuperGLUE und ressourcenarmen Aufgaben.

Unser letzter Beitrag besteht darin, PERFECT vorzuschlagen, eine einfache und effiziente Methode zur Feinabstimmung von PLMs in wenigen Schritten *ohne sich auf Handarbeit zu verlassen*, die bei nur 32 Datenpunkten sehr effektiv ist. Dies steht im Gegensatz zu den aktuellen Methoden zur Feinabstimmung von vortrainierten maskierten Sprachmodellen (PLM) in wenigen Beispielen, die sorgfältig konstruierte Eingabeaufforderungen und Verbalisierer für jede neue Aufgabe erfordern, um Beispiele in ein Lückentextformat umzuwandeln, das das PLM bewerten kann. PERFECT trifft zwei wichtige Designentscheidungen: Erstens zeigen wir, dass manuell erstellte Task-Prompts durch *task-spezifische Adapter* ersetzt werden können, die eine dateneffiziente Feinabstimmung ermöglichen und die Arbeitsspeicher- und Festplattenspeicherkosten ungefähr um den Faktor 5 beziehungsweise 100 reduzieren. Zweitens lernen wir, anstatt handgefertigte Verbalisierer zu verwenden, während der Feinabstimmung ein neues *Multi-Token-Label-Einbetten*, das nicht an das Modellvokabular gebunden ist und uns erlaubt, komplexe autoregressive Dekodierung zu vermeiden. Diese Einbettungen sind nicht nur aus begrenzten Daten erlernbar, sondern ermöglichen auch fast 100-mal schnelleres Training und Schlussfolgerungen. Experimente mit einer breiten Palette von NLP-Aufgaben mit wenigen

Beispielen zeigen, dass PERFECT, obwohl es einfach und effizient ist, auch bestehende hochmoderne Lernmethoden mit wenigen Beispielen übertrifft.

Stichworte: Transferlernen, Verallgemeinerung, Feinabstimmung, Bias-Reduktion, Multitask-Lernen, Lernen mit wenigen Beispielen, ressourcenarme Aufgaben, Parameter-effiziente Feinabstimmung, Adapter, Robustheit.

Acknowledgements

This thesis is the result of the support of several people, to whom I am extremely grateful. First and foremost, I am deeply grateful to my supervisors Prof. James Henderson and Prof. Volkan Cevher. I have been exceptionally fortunate to have my supervisors, who were truly among the most brilliant people I have met in my life, but not only this, they were among the most supportive people I could have ever imagined to have the opportunity to work with. This was an absolute pleasure and joy for me to do this journey with them. I would like to thank them both deeply for continuously encouraging my research and for allowing me to grow as a research scientist. This experience is invaluable for my career.
I would like to thank the jury members of my thesis: Prof. Luke Zettlemoyer, Dr. Charles Blundell, Prof. Edouard Bugnion, and Prof. Antoine Bosselut, for their time reviewing my thesis, and for joining and evaluating my oral exam. I would also like to thank Swiss National Science Foundation for funding my research through the LAOS project, which helped me pursue a curiosity-driven research agenda.
I am deeply indebted to Charles Blundell who provided me with an invaluable opportunity to do an internship in his team at DeepMind. My gratitude also goes to my wonderful mentors Andrea Banino and Tim Scholtes, who were extremely supportive and amazing people to work with. I am very thankful to them all for creating this enjoyable experience for me.
I am truly grateful to Sebastian Ruder from Google Research, who was among the most brilliant and supportive people I had the chance to work with. I am truly grateful to my hosting team at Google Research, AI pioneer Geoff Hinton's group, for providing me with the opportunity to work with them. In particular, I would like to thank Mostafa Dehghani for continuously encouraging me and aspiring me through my internship.
My gratitude also goes to my hosting team at the Meta AI and Meta AI Research, particularly my amazing manager and wonderful peers, Majid Yazdani, Luke Zettlemoyer, Lambert Mathias, Marzieh Saeidi, and Ves Stoyanov for giving me a fantastic internship opportunity and the freedom to explore challenging research directions and for being truly supportive. This was an invaluable opportunity and an immensely enjoyable experience for me to work with them.
I am truly grateful to all my colleagues in the Natural language processing group at Idiap. I appreciate all the stimulating and fruitful discussions, which have inspired and motivated me to improve my research. I especially thank Florian Mai, Suraj Srinivas, and Apoorv Vyas. I am indebted for their support and insightful discussions we had which substantially helped me through my PhD. Additionally, my sincere gratitude goes to all my friends at Idiap and EPFL.
Last but not least, I would like to thank my family members who have always been truly supportive

and caring to me.

Martigny, February 3, 2023 Rabeeh Karimi Mahabadi

Publications Based on the Thesis

Chapter 2 of this thesis is based on:

- R. Karimi Mahabadi & Y. Belinkov & J. Henderson, "End-to-End Bias Mitigation by Modelling Biases in Corpora", ACL, 2020.

Chapter 3 of this thesis is based on:

- R. Karimi Mahabadi & Y. Belinkov & J. Henderson, "Variational Information Bottleneck for Effective Low-Resource Fine-Tuning", ICLR, 2021.

Chapter 4 of this thesis is based on:

- R. Karimi Mahabadi & S. Ruder & M. Dehghani & J. Henderson, "Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks", ACL, 2021. [**Oral Presentation**]

Chapter 5 of this thesis is based on:

- R. Karimi Mahabadi & J. Henderson & S. Ruder, "Compacter: Efficient Low-Rank Hypercomplex Adapter Layers", NeurIPS, 2021.

Chapter 6 of this thesis based on:

- R. Karimi Mahabadi, L. Zettlemoyer, J. Henderson, L.Mathias, M. Saeidi, V. Stoyanov, M. Yazdani, "Prompt-free and Efficient Few-shot Learning with Language Models", ACL, 2022.

0.1 Other Publications

I additionally had the opportunity to contribute to the following papers during my PhD studies:

- "Learning-Based Compressive MRI", B. Gözcü, R. Karimi Mahabadi, Y. Li, E. Ilıcak, T. Cukur, J. Scarlett, and V.Cevher, IEEE Transactions on Medical Imaging, 2017.
- "A Learning-Based Framework for Quantized Compressed Sensing", R. Karimi Mahabadi, J. Lin, V. Cevher, IEEE Signal Processing Letters, 2018.
- "Real-time DCT Learning-based Reconstruction of Neural Signals", R. Karimi Mahabadi, C. Aprile, V. Cevher, EUSIPCO, 2018.

Publications Based on the Thesis

- "A Non-Euclidean Gradient Descent Framework for Non-Convex Matrix Factorization", Y. Hsieh, Y. Kao, R. Karimi Mahabadi, A. Kyrillidis, and V. Cevher, IEEE Transactions on Signal Processing, 2018.

- "ParsiNLU: A Suite of Language Understanding Challenges for Persian", Daniel Khashabi, Arman Cohan, Siamak Shakeri, Pedram Hosseini, Pouya Pezeshkpour, Malihe Alikhani, Moin Aminnaseri, Marzieh Bitaab, Faeze Brahman, Sarik Ghazarian, Mozhdeh Gheini, Arman Kabiri, Rabeeh Karimi Mahabadi, Omid Memarrast, Ahmadreza Mosallanezhad, Erfan Noury, Shahab Raji, Mohammad Sadegh Rasooli, Sepideh Sadeghi, Erfan Sadeqi Azer, Niloofar Safi Samghabadi, Mahsa Shafaei, Saber Sheybani, Ali Tazarv, Yadollah Yaghoobzadeh, TACL, (presented in EMNLP), 2021.

Contents

List of Figures

List of Tables

1 Introduction & Background

I do what I feel is right. I am not
scared to walk on the new path and take risk.

Aamir Khan

Natural language processing (NLP), i.e., allowing machines to break down and understand human language, is a complicated task requiring training neural models capable of grasping a variety of knowledge. This knowledge can range from low-level, like syntactic rules or interpreting the meaning of individual words, to high-level understanding, such as semantic plausibility, e.g. that elephants do not fit in a fridge requires prior knowledge about elephants and fridges. There is almost always not sufficient data available to learn these different types of required knowledge from task data. This necessitates training neural language models (Bengio et al., 2003) which can build a general understanding of the text while being able to get adapted to downstream tasks. Early successes in this goal were found in learning word embeddings (Mikolov et al., 2013b,a; Pennington et al., 2014) to map words to a space where similar words obtain similar vector representations. This acts as a single layer of representation, which is then fed to task-specific architectures. Later efforts, use a Recurrent neural network (RNN) with multiple layers of representation to obtain more powerful contextualized representations (Dai and Le, 2015; McCann et al., 2017; Peters et al., 2018) while still feeding them to task-specific models. More recently, this paradigm shifted to pretraining the entire transformer (Vaswani et al., 2017) and recurrent models, followed by fine-tuning on downstream tasks. This flexible technique subsequently eliminates the need for learning task-specific models (Radford et al., 2018; Devlin et al., 2019; Howard and Ruder, 2018), becoming a dominating technique in NLP and computer vision.

Broadly speaking, pretraining is used to train an entire neural model on a large amount of data, to learn the general-purpose knowledge and understanding of language, which can later be adapted to each individual downstream task, referred to as transfer learning. In computer vision, transfer learning (Oquab et al., 2014; Jia et al., 2014; Huh et al., 2016; Yosinski et al., 2014) is usually performed by pretraining models on large labeled datasets such as ImageNet (Russakovsky et al., 2015; Deng et al., 2009). In NLP, however, pretraining is usually done through unsupervised objectives such as predicting the next

token given previous tokens auto-regressively (Brown et al., 2020), or replacing random tokens with a [MASK] token and predicting them given their context (Devlin et al., 2019), masking random contiguous spans rather than individual tokens and predicting the span using the tokens at the boundary (Joshi et al., 2020), and corrupting the text with an arbitrary noising function and learning to reconstruct the original text (Lewis et al., 2020), to name a few. Unsupervised pretraining objectives without requiring labeled data allow leveraging a massive amount of data for pretraining available through the Internet. For instance, the Common Crawl project[1] provides 20TB of data every month. This combines with the fact the language models scale remarkably well, allowing obtaining striking performance as pretraining larger-scale models on larger dataset sizes (Chowdhery et al., 2022; Du et al., 2022; Brown et al., 2020; Hoffmann et al., 2022; Hestness et al., 2017; Shazeer et al., 2017; Jozefowicz et al., 2016; Mahajan et al., 2018; Radford et al., 2019; Shazeer et al., 2018; Huang et al., 2018; Keskar et al., 2019).

The success of pretrained language models (PLMs) on a wide variety of tasks (Raffel et al., 2020; Chowdhery et al., 2022; Du et al., 2022; Brown et al., 2020; Hoffmann et al., 2022), has given rise to a broad spectrum of research efforts such as developing new pretraining objectives (Dai and Le, 2015; Ramachandran et al., 2017; Radford et al., 2018; Devlin et al., 2019; Yang et al., 2019; Liu et al., 2019a; Wang et al., 2019a; Song et al., 2019; Dong et al., 2019; Joshi et al., 2020), collecting new benchmarks to pose a more rigorous test of language understanding (Wang et al., 2019b), efficient fine-tuning methods (Mahabadi et al., 2021a,b; Howard and Ruder, 2018; Houlsby et al., 2019; Peters et al., 2019), collecting more clean unlabeled datasets (Raffel et al., 2020), improving the efficiency of language models (Dettmers et al., 2022; Rajbhandari et al., 2019; Smith et al., 2022), and many others.

1.1 Open problems

In spite of this rapid progress, transfer learning and training neural models which can generalize to unseen domains is a multifaceted problem, requiring addressing several open questions. In this thesis, we look at a) training models robust to biases in datasets b) reducing overfitting, especially in low-source settings c) learning from multiple resources and generalization to unseen domains d) efficient fine-tuning methods e) few-shot learning with PLMs:

- *Training models robust to biases in datasets* Several recent studies have shown that neural models (Devlin et al., 2019; Radford et al., 2018) tend to rely on unwanted dataset biases[2], and leverage superficial correlations between the label and existing shortcuts in the training dataset to perform surprisingly well, without learning the underlying task (Kaushik and Lipton, 2018; Gururangan et al., 2018; Poliak et al., 2018; Schuster et al., 2019; McCoy et al., 2019b), resulting in models that fail to generalize to out-of-domain datasets (i.e. when the data provided to the model at test time is significantly different from what it is trained on) and are likely to perform poorly in real-world scenarios. For instance, natural language inference (NLI) is supposed to test the ability of a model to determine whether a hypothesis sentence (*There is*

[1] `http://commoncrawl.org`

[2] We use biases, heuristics or shortcuts interchangeably.

no teacher in the room) can be inferred from a premise sentence (*Kids work at computers with a teacher's help*) (Dagan et al., 2005).[3] However, recent work has demonstrated that large-scale NLI benchmarks contain annotation artifacts; certain words in the hypothesis that are highly indicative of inference class and allow models that do not consider the premise to perform unexpectedly well (Poliak et al., 2018; Gururangan et al., 2018). As an example it has been shown that negation words such as "nobody", "no", and "not" in the hypothesis are often highly correlated with the contradiction label. It is, therefore, crucial to develop techniques to reduce the reliance on dataset biases during the training of neural models.

- *Reducing overfitting especially on low-resource setting* Pretrained language models have a huge number of parameters, potentially making fine-tuning susceptible to overfitting in a low-resource setting, i.e. when there is a limited amount of training data available for training. In particular, the task-universal nature of large-scale pretrained sentence representations means that much of the information in these representations is irrelevant to a given target task. If the amount of target task data is small, it can be hard for fine-tuning to distinguish relevant from irrelevant information, leading to overfitting on statistically spurious correlations between the irrelevant information and target labels. It is therefore crucial to address the problem of overfitting and propose fine-tuning methods to improve transfer learning in low-resource scenarios. Addressing learning from low-resource tasks is especially an important topic in NLP (Cherry et al., 2019) because data annotation is costly and time-consuming, and in several tasks access to data is limited.

- *Learning from multiple resources and generalizing to unseen domains* Multi-task learning with pretrained language models (Ruder, 2017) is appealing for multiple reasons: 1) Training individual models per task results in higher computational costs, which hinders their deployment and maintenance. These costs are substantially reduced by training a single model. 2) Fine-tuning the model across multiple tasks allows sharing information between the different tasks and positive transfer to other related tasks. Specifically, when target datasets have limited training data, multi-task learning improves the performance compared to individually trained models (Liu et al., 2019a; Ratner et al., 2018). However, multi-task fine-tuning can result in models underperforming on high-resource tasks due to constrained capacity (Arivazhagan et al., 2019; McCann et al., 2018). Additionally, such models can be susceptible to *task interference* or *negative transfer*, where achieving good performance on one task can hinder performance on another (Wang et al., 2019d). It is therefore important to be able to fine-tune PLMs on multiple tasks in a way that information across tasks can be shared while avoiding negative task interference, and ideally enabling transferring this knowledge to unseen domains.

- *Efficient fine-tuning methods* PLMs are generally applied to downstream tasks via fine-tuning (Howard and Ruder, 2018), which requires updating *all* parameters and storing one copy of the fine-tuned model per task. This causes substantial storage and deployment costs and hinders the applicability of large-scale PLMs to real-world applications. Additionally, fine-tuning of over-parameterized models on low-resource datasets has been shown to be subject to instabilities and may lead to poor performance (Peters et al., 2019; Dodge et al., 2020). This is therefore pivotal

[3] The given sentences are in the contradictory relation, and the hypothesis cannot be inferred from the premise.

to develop practical, memory-efficient methods that train a minimum set of parameters while achieving performance on par or better than full fine-tuning for state-of-the-art NLP models.

- *Few-shot learning with PLMs* Recently, GPT-3 (Brown et al., 2020) has obtained impressive few-shot performance on several natural understanding tasks. GPT3 conditions on a *language prompt* and a *few demonstrations of the task* without updating the weights of the underlying model. However, it has over 175B parameters, hindering its application in real-world scenarios. This is therefore of huge practical value, to obtain similar or better few-shot performance on much smaller PLMs like BERT (Devlin et al., 2019) or RoBERTa (Liu et al., 2019b). This not only allows an efficient training/inference with a low compute budget, but also reduces the need for costly annotated examples, and is practical as this is easy to annotate a few training examples (e.g., 32 samples).

In the rest of this chapter, we shall provide background relevant to the topics of this thesis, which we will use to state our main contributions at the end of this chapter.

1.2 Background & Related work

1.2.1 Bias reduction

As a result of the existence of dataset biases, models exploiting statistical shortcuts during training often perform poorly on out-of-domain datasets, especially if the datasets are carefully designed to limit the spurious cues. To allow proper evaluation, recent studies have tried to create new evaluation datasets that do not contain such biases (Gururangan et al., 2018; Schuster et al., 2019; McCoy et al., 2019b). Unfortunately, it is hard to avoid spurious statistical cues in the construction of large-scale benchmarks, and collecting new datasets is costly (Sharma et al., 2018). It is pivotal to develop techniques to reduce the reliance on biases during the training of neural models.

To address dataset biases, researchers have proposed to augment datasets by balancing the existing cues (Schuster et al., 2019) or to create an adversarial dataset (Jia and Liang, 2017). However, collecting new datasets, especially at a large scale, is costly, and thus remains an unsatisfactory solution. It is, therefore, crucial to develop strategies to allow models to be trained on the existing biased datasets. Schuster et al. (2019) propose to first compute the n-grams in the dataset's claims that are the most associated with each fact-verification label. They then solve an optimization problem to assign a balancing weight to each training sample to alleviate the biases. However, this model does not work end-to-end, which is convenient in practice. Additionally, Belinkov et al. (2019a) propose adversarial techniques to remove from the NLI sentence encoder the features that allow a hypothesis-only model to succeed. However, in general, the features used by the hypothesis-only model can include some information necessary to perform the NLI task, and removing such information from the sentence representation can hurt the performance of the full model. Their approach consequently degrades the performance on the hard SNLI set, which is expected to be less biased. However, this is valuable to developing debiasing strategies that can improve the generalization of neural models, while not hurting their in-domain performance.

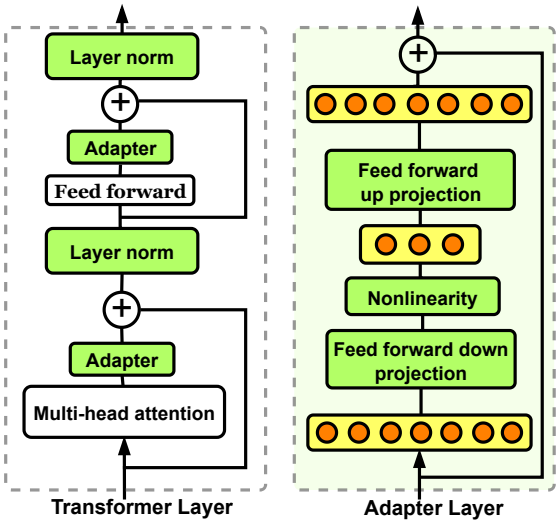

Figure 1.1: Left: Adapter integration in a PLM. Right: An adapter architecture. Adapters are usually inserted after the feed-forward and self-attention modules. During training, only the green components are optimized.

1.2.2 Reducing overfitting on low-resource setting

Several regularization techniques have been used to reduce over-fitting when fine-tuning PLMs, especially on low-resource settings: 1) *Dropout:* (Srivastava et al., 2014), a widely used stochastic regularization technique used in multiple large-scale language models (Devlin et al., 2019; Yang et al., 2019; Vaswani et al., 2017) to mitigate overfitting. 2) *Mixout* (Lee et al., 2019): is a stochastic regularization technique inspired by Dropout with the goal of preventing catastrophic forgetting during fine-tuning. Mixout regularizes the learning to minimize the deviation of a fine-tuned model from the pretrained initialization. It replaces the model parameters with the corresponding value from the pretrained model with probability p. 3) *Weight Decay (WD)*: is a common regularization technique to improve generalization (Krogh and Hertz, 1992). It regularizes the large weights w by adding a penalization term $\frac{\lambda}{2}\|w\|$ to the loss, where λ is a hyperparameter specifying the strength of regularization. Chelba and Acero (2004) and Daumé III (2007) adapt WD for fine-tuning of the pretrained models, and propose to replace this regularization term with $\lambda\|w - w_0\|$, where w_0 are the weights of the pretrained models. Recently, Lee et al. (2019) demonstrated that the latter formulation of WD works better for fine-tuning of BERT than conventional WD and can improve generalization on small training sets. In another line of work, Phang et al. (2018) proposed to perform an extra data-rich intermediate supervised task pretraining followed by fine-tuning on the target task. They showed that their method leads to improved fine-tuning performance on the GLUE benchmark (Wang et al., 2019c). However, their method requires pretraining with a large intermediate task. However, this is of practical value, to address overfitting by leveraging only the provided low-resource target datasets.

1.2.3 Efficient Fine-tuning of PLMs

Adapter layers Recent work has shown that fine-tuning *all* parameters of PLMs with a large number of parameters in low-resource datasets can lead to a sub-optimal solution (Peters et al., 2019; Dodge et al., 2020). As shown in Figure 1.1, Rebuffi et al. (2018) and Houlsby et al. (2019) suggest an efficient alternative, by inserting small task-specific modules called *adapters* within layers of a PLMs. They then only train the newly added adapters and layer normalization, while fixing the remaining parameters of a PLM.

Each layer of a transformer model is composed of two primary modules: a) an attention block, and b) a feed-forward block, where both modules are followed by a skip connection. As depicted in Figure 1.1, adapters are normally inserted after each of these blocks before the skip connection. Adapters are bottleneck architectures. By keeping input and output dimensions the same, they introduce no additional architectural changes. Each adapter, $A(.) \in \mathbb{R}^H$, consists of a down-projection, $D(.) \in \mathbb{R}^{H \times B}$, a non-linearity, such as GeLU (Hendrycks and Gimpel, 2016), and an up-projection $U(.) \in \mathbb{R}^{B \times H}$, where H is the dimension of input hidden states $\boldsymbol{x}$, and B is the bottleneck size. Formally defined as:

$$A(\boldsymbol{x}) = U(\text{GeLU}(D(\boldsymbol{x}))) + \boldsymbol{x}, \tag{1.1}$$

Parameter-efficient fine-tuning methods Li et al. (2018) and Aghajanyan et al. (2021) study training models in a low-dimensional randomly oriented subspace instead of their original parameter space. Another recent line of work has shown that pretrained models such as BERT are redundant in their capacity, allowing for significant sparsification without much degradation in end metrics (Chen et al., 2020; Prasanna et al., 2020; Desai et al., 2019). Such methods, however, remain not well supported by current hardware and often perform worse compared to dedicated efficient architectures (Blalock et al., 2020). Cai et al. (2020) propose to freeze the weights and only train the biases. By not storing intermediate activations, this method enables substantial memory savings. Ravfogel et al. (2021) study a similar method for PLMs that fine-tunes only the biases and the final output layer. Other work includes prompt tuning (Lester et al., 2021) which is the successor variant of Li and Liang (2021), which prepends a randomly initialized continuous prompt to the input. Another variant of it initializes prompts using token embeddings of the pretrained language model's vocabulary (Lester et al., 2021).

Efficient fine-tuning methods in the multi-task learning setting Multi-task learning, i.e., learning a unified model to perform well on multiple different tasks, is a challenging problem in NLP. It requires addressing multiple challenges such as catastrophic forgetting, and handling disproportionate task sizes resulting in a model overfitting in low-resource tasks while underfitting in high-resource ones (Arivazhagan et al., 2019). Liu et al. (2019a) proposed Multi-Task Deep Neural Network (MTDNN) for learning from multiple NLU tasks. Although MTDNN obtains impressive results on GLUE, it applies multi-task learning as a form of pretraining followed by task-specific fine-tuning. In another line of research, Clark et al. (2019c) proposed to learn multi-task models with knowledge distillation. Houlsby

et al. (2019) trained adapters for each task separately, keeping the model fixed. Stickland and Murray (2019) share the model parameters across tasks and introduce task-specific adapter parameters. In another approach, Oswald et al. (2020) proposed a task-conditioned hypernetwork to generate all the weights of the target model in a continual learning setup, where tasks are learned sequentially. Similarly, Jin et al. (2020) generate the full model from task-specific descriptions in different domains. Note that such methods dealing with fully generating all the weights of the target models are not efficient. Prior work also proposed meta-learning or Bayesian approaches to generate softmax layer parameters for new settings (Bansal et al., 2020; Ponti et al., 2020). Meta-learning approaches are notoriously slow to train. In addition, generating softmax parameters requires a substantially higher number of parameters, leaves the method unable to adapt the lower layers of the model, and restricts their application to classification tasks. This is of paramount importance to develop efficient fine-tuning methods for multi-task learning settings, which allow the model to generalize to the new unseen domains.

1.2.4 Few-shot Learning with Pretrained Language Models

In this section, we review the related work on few-shot learning with pretrained language models.

Standard Fine-tuning In standard fine-tuning with PLMs (Devlin et al., 2019), first a special [CLS] token is appended to the input $\boldsymbol{x}$, and then the PLM maps it to a sequence of hidden representations $\boldsymbol{h} = (\boldsymbol{h_1}, ..., \boldsymbol{h_S})$ with $\boldsymbol{h_i} \in \mathbb{R}^H$, where H is the hidden dimension, and S is the maximum sequence length. Then, a classifier, *softmax*$(\boldsymbol{W^T h}_{\text{[CLS]}})$, using the embedding of the classification token ($\boldsymbol{h}_{\text{[CLS]}}$), is trained end-to-end for each downstream task. The main drawback of this approach is the discrepancy between the pre-training and fine-tuning phases since PLMs have been trained to *predict mask tokens* in a masked language modeling task (Devlin et al., 2019).

Prompt-based tuning To address this discrepancy, *prompt-based fine-tuning* (Schick and Schütze, 2021a,b; Gao et al., 2021) formulates tasks in a cloze-format (Taylor, 1953). This way, the model can predict targets with a *masked language modeling (MLM) objective*. For example, as shown in Figure 1.2, for a sentiment classification task, inputs are converted to:

$$\boldsymbol{x}_{\text{prompt}} = \text{[CLS]}\ \boldsymbol{x}\ .\ \underbrace{\text{It was}}_{\text{pattern}}\ \text{[MASK]}\ .\ \text{[SEP]}$$

Then, the PLM determines which *verbalizer* (e.g., 'great' and 'terrible') is the most likely substitute for the mask in the $\boldsymbol{x}_{\text{prompt}}$. This subsequently determines the score of targets ('positive' or 'negative'). Then, a mapping, $\mathcal{M}: \mathcal{Y} \rightarrow \mathcal{V}$, from target labels to individual words in a PLM's vocabulary is learned. We refer to this mapping as *verbalizers*. Then the input is converted to $\boldsymbol{x}_{\text{prompt}} = \mathcal{T}(\boldsymbol{x})$ by appending a *pattern* and a *mask token* to $\boldsymbol{x}$ so that it has the format of a masked language modeling input. Then, the classification task is converted to an MLM objective (Tam et al., 2021; Schick and Schütze, 2021a),

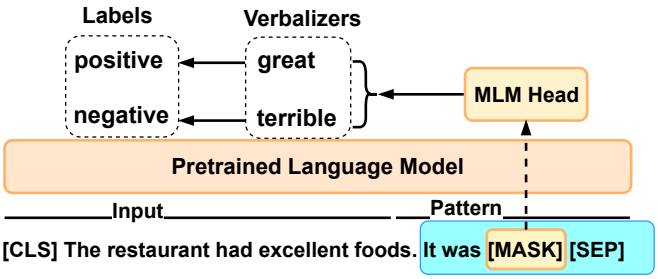

Figure 1.2: Existing few-shot fine-tuning methods rely on handcrafted patterns and verbalizers to convert tasks to a masked language modeling format.

and the PLM computes the probability of the label y as:

$$p(y|\boldsymbol{x}) = p([\text{MASK}] = \mathcal{M}(y)|\boldsymbol{x}_{\text{prompt}}) = \frac{\exp(\boldsymbol{W}_{\mathcal{M}(y)}^T \boldsymbol{h}_{[\text{MASK}]})}{\sum_{v' \in \mathcal{V}} \exp(\boldsymbol{W}_{v'}^T \boldsymbol{h}_{[\text{MASK}]})}, \tag{1.2}$$

where $\boldsymbol{h}_{[\text{MASK}]}$ is the last hidden representation of the mask, and $\boldsymbol{W}_v$ shows the output embedding of the PLM for each verbalizer $v \in \mathcal{V}$. For many tasks, verbalizers have multiple tokens. Schick and Schütze (2021b) extended (1.2) to multiple mask tokens by adding the maximum number of mask tokens M needed to express the outputs (verbalizers) for a task. In that case, Schick and Schütze (2021b) computes the probability of each class as the summation of the log probabilities of each token in the corresponding verbalizer, and then they add a hinge loss to ensure a margin between the correct verbalizer and the incorrect ones.

This formulation obtained impressive few-shot performance with PLMs. However, the success of this approach heavily relies on engineering handcrafted *patterns* and *verbalizers*. Coming up with suitable verbalizers and patterns can be difficult (Mishra et al., 2021). Additionally, the performance is sensitive to the wording of patterns (Zhao et al., 2021; Perez et al., 2021; Schick and Schütze, 2021a; Jiang et al., 2020) or to the chosen verbalizers (Webson and Pavlick, 2021).

Researchers continuously tried to address the challenges of manually engineered patterns and verbalizers: a) Learning the patterns in a continuous space (Li and Liang, 2021; Qin and Eisner, 2021; Lester et al., 2021), while freezing PLM for efficiency, has the problem that, in most cases, such an approach only works with very large scale PLMs (Lester et al., 2021), and lags behind full fine-tuning in a general setting while being inefficient and not as effective compared to adapters (Mahabadi et al., 2021a). b) Optimizing patterns in a discrete space (Shin et al., 2020; Jiang et al., 2020; Gao et al., 2021) has the problem that such methods are computationally costly. c) Automatically finding verbalizers in a discrete way Schick et al. (2020); Schick and Schütze (2021a) is computationally expensive and does not perform as well as manually designed ones. d) Removing manually designed patterns (Logan IV et al., 2021) substantially lags behind the expert-designed ones. This is therefore pivotal to propose few-shot learning methods which do not rely on any handcrafted patterns and verbalizers.

1.3 Research Questions & Contributions

Having discussed the relevant background material, we are now ready to delve into the concrete research questions we consider in this thesis.

Research Question 1 *Is it possible to train robust natural language understanding (NLU) models that are not relying on unwanted dataset* biases *and generalize better to out-of-domain datasets?*

We answer this research question in Chapter 2 by proposing two learning strategies to train neural models, which are more robust to dataset biases and transfer better to out-of-domain datasets. The biases are specified in terms of one or more *bias-only models*, which learn to leverage the dataset biases. During training, the bias-only models' predictions are used to adjust the loss of the base model to reduce its reliance on biases by down-weighting the biased examples and focusing training on the *hard* examples. We experiment on large-scale natural language inference and fact verification benchmarks, evaluating on out-of-domain datasets that are specifically designed to assess the robustness of models against known biases in the training data. Results show that our debiasing methods greatly improve robustness in all settings and better transfer to other textual entailment datasets.

Research Question 2 *Is it possible to reduce the overfitting of large-scale pretrained language models when fine-tuned especially on low-resource scenarios and suppress the irrelevant features, learned during pretraining, for a given target task?*

In Chapter 3 we take a bird's eye view on fine-tuning large-scale pretrained language models on low-resource datasets and ask how can we reduce overfitting when the number of training data is limited. We start with the observation that due to the task-universal nature of pretraining of language models, such models are general-purpose feature extractors and many of these features are inevitably irrelevant for a given target task. If the amount of target task data is small, it can be hard for fine-tuning to distinguish relevant from irrelevant information, leading to overfitting on statistically spurious correlations between the irrelevant information and target labels. Crucially, learning low-resource tasks is an important topic in NLP (Cherry et al., 2019) because annotating more data can be very costly and time-consuming, and because in several tasks access to data is limited. This leaves an open question of how we can suppress these irrelevant features while keeping the most concise representation which can still solve the task. To address this problem, we propose to use Variational Information Bottleneck (VIB) (Alemi et al., 2017) to suppress irrelevant features when fine-tuning on low-resource target tasks and show that our method successfully reduces overfitting. Moreover, we show that our VIB model finds sentence representations that are more robust to biases in natural language inference datasets, and thereby substantially improves generalization to out-of-domain datasets. Evaluation on seven low-resource datasets in different tasks shows that our method significantly improves transfer learning in low-resource scenarios, surpassing prior work. Moreover, it improves generalization on 13 out of 15 out-of-domain natural language inference benchmarks.

Research Question 3 *Is it possible to fine-tune pretrained language models across multiple tasks efficiently, allowing sharing information across tasks while eliminating negative task interference?*

State-of-the-art parameter-efficient fine-tuning methods rely on introducing adapter modules between the layers of a pretrained language model. However, such modules are trained separately for each task and thus do not enable sharing information across tasks. We ask if there can be an effective and parameter-efficient way to share information across multiple adapters to enable positive transfer to low-resource and related tasks. To address this problem and to enable sharing information across tasks while reaping the benefits of adapter layers, in Chapter 4, we propose HYPERFORMER++, which employs a compact hypernetwork (Ha et al., 2017; Oswald et al., 2020) shared across tasks and layers. The hypernetwork is a network that generates the weights of another network. The hypernetwork then learns to generate task and layer-specific adapter parameters, conditioned on task and layer id embeddings in a transformer model. This parameter-efficient multi-task learning framework allows us to achieve the best of both worlds by sharing knowledge across tasks via hypernetworks while enabling the model to adapt to each individual task through task-specific adapters. Experiments on the well-known GLUE benchmark show improved performance in multi-task learning while adding only 0.29% parameters per task. We additionally demonstrate substantial performance improvements in few-shot domain generalization across a variety of tasks.

Research Question 4 *Is it possible to fine-tune pretrained language models in a parameter-efficient and stable way in a low-resource setting?*

We answer this question in Chapter 5 by proposing COMPACTER, a method for fine-tuning large-scale language models with a better trade-off between task performance and the number of trainable parameters than prior work. COMPACTER accomplishes this by building on top of ideas from adapters, low-rank optimization (Aghajanyan et al., 2021), and parameterized hypercomplex multiplication layers (Zhang et al., 2021a). Specifically, COMPACTER inserts task-specific weight matrices into a pretrained model's weights, which are computed efficiently as a sum of Kronecker products between shared "slow" weights and "fast" rank-one matrices defined per COMPACTER layer. By only training 0.047% of a pretrained model's parameters, COMPACTER performs on par with standard fine-tuning on GLUE and outperforms standard fine-tuning on SuperGLUE (Wang et al., 2019b) and low-resource settings.

Research Question 5 *Is it possible to few-shot fine-tune pretrained language models* without relying on any handcrafting*?*

Current methods for few-shot fine-tuning of pretrained masked language model (PLM) require carefully engineered prompts and verbalizers for each new task, to convert examples into a cloze-format that the PLM can score. In Chapter 6, we propose PERFECT, a simple and efficient method for few-shot fine-tuning of PLMs *without relying on any such handcrafting*, which is highly effective given as few as 32 data points. PERFECT makes two key design choices: First, we show that manually engineered task prompts can be replaced with *task-specific adapters* that enable sample-efficient fine-tuning and reduce memory and storage costs by roughly factors of 5 and 100, respectively. Second, instead of

using handcrafted verbalizers, we learn a new *multi-token label embedding* during fine-tuning which is not tied to the model vocabulary and which allows us to avoid complex auto-regressive decoding. These embeddings are not only learnable from limited data but also enable nearly 100x faster training and inference. Experiments on a wide range of few-shot NLP tasks demonstrate that PERFECT, while being simple and efficient, also outperforms existing state-of-the-art few-shot learning methods.

2 End-to-End Bias Mitigation by Modelling Biases in Corpora

Several recent studies have shown that strong natural language understanding (NLU) models are prone to relying on unwanted dataset *biases* without learning the underlying task, resulting in models that fail to generalize to out-of-domain datasets and are likely to perform poorly in real-world scenarios. In this Chapter, we propose two learning strategies to train neural models, which are more robust to such biases and transfer better to out-of-domain datasets. The biases are specified in terms of one or more *bias-only models*, which learn to leverage the dataset biases. During training, the bias-only models' predictions are used to adjust the loss of the base model to reduce its reliance on biases by down-weighting the biased examples and focusing training on the *hard* examples. We experiment on large-scale natural language inference and fact verification benchmarks, evaluating on out-of-domain datasets that are specifically designed to assess the robustness of models against known biases in the training data. Results show that our debiasing methods greatly improve robustness in all settings and better transfer to other textual entailment datasets.[1]

2.1 Introduction

Recent neural models (Devlin et al., 2019; Radford et al., 2018; Chen et al., 2017) have achieved high and even near human-performance on several large-scale natural language understanding benchmarks. However, it has been demonstrated that neural models tend to rely on existing idiosyncratic biases in the datasets, and leverage superficial correlations between the label and existing shortcuts in the training dataset to perform surprisingly well, without learning the underlying task (Kaushik and Lipton, 2018; Gururangan et al., 2018; Poliak et al., 2018; Schuster et al., 2019; McCoy et al., 2019b). For instance, natural language inference (NLI) is supposed to test the ability of a model to determine whether a hypothesis sentence (*There is no teacher in the room*) can be inferred from a premise sentence (*Kids work at computers with a teacher's help*) (Dagan et al., 2005).[2] However, recent work has demonstrated that large-scale NLI benchmarks contain annotation artifacts; certain words in the hypothesis that are highly indicative of inference class and allow models that do not consider the

[1] Our code and data are publicly available in `https://github.com/rabeehk/robust-nli`.

[2] The given sentences are in the contradictory relation, and the hypothesis cannot be inferred from the premise.

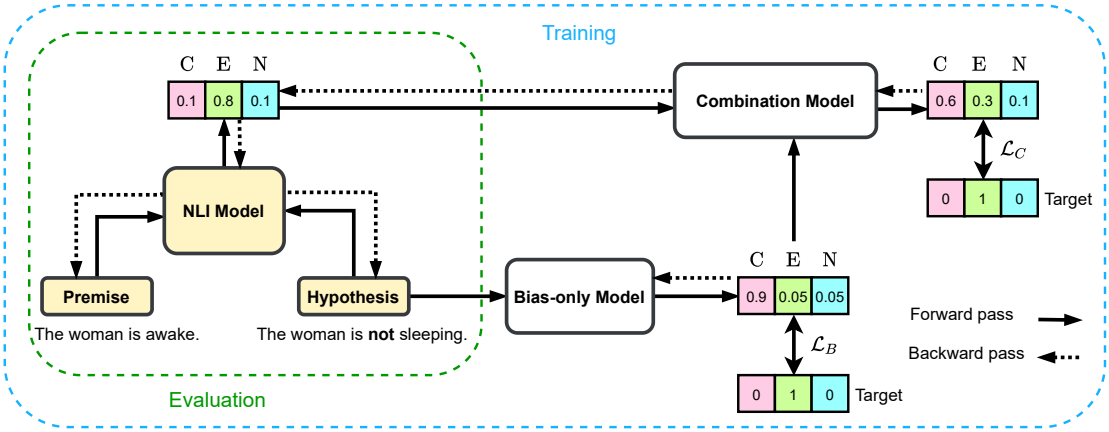

Figure 2.1: An illustration of our debiasing strategies applied to an NLI model. The bias-only model only sees the hypothesis, where negation words like "not" are highly correlated with the contradiction label. We train a robust NLI model by training it in combination with the bias-only model and motivate it to learn different strategies than the ones used in the bias-only model. The robust NLI model does not rely on the shortcuts and obtains improved performance on the test set.

premise to perform unexpectedly well (Poliak et al., 2018; Gururangan et al., 2018). As an example, in some NLI benchmarks, negation words such as "nobody", "no", and "not" in the hypothesis are often highly correlated with the contradiction label.

As a result of the existence of such biases, models exploiting statistical shortcuts during training often perform poorly on out-of-domain datasets, especially if the datasets are carefully designed to limit the spurious cues. To allow proper evaluation, recent studies have tried to create new evaluation datasets that do not contain such biases (Gururangan et al., 2018; Schuster et al., 2019; McCoy et al., 2019b). Unfortunately, it is hard to avoid spurious statistical cues in the construction of large-scale benchmarks, and collecting new datasets is costly (Sharma et al., 2018). It is, therefore, crucial to develop techniques to reduce the reliance on biases during the training of the neural models.

We propose two end-to-end debiasing techniques that can be used when the existing bias patterns are identified. These methods work by adjusting the cross-entropy loss to reduce the biases learned from the training dataset, down-weighting the biased examples so that the model focuses on learning the hard examples. Figure 2.1 illustrates an example of applying our strategy to prevent an NLI model from predicting the labels using existing biases in the hypotheses, where the bias-only model only sees the hypothesis. Our strategy involves adding this bias-only branch f_B on top of the base model f_M during training. We then compute the combination of the two models f_C in a way that motivates the base model to learn different strategies than the ones used by the bias-only branch f_B. At the end of the training, we remove the bias-only classifier and use the predictions of the base model.

In our first proposed method, Product of Experts, the training loss is computed on an ensemble of the base model and the bias-only model, which reduces the base model's loss for the examples that the bias-only model classifies correctly. For the second method, Debiased Focal Loss, the bias-only

predictions are used to directly weight the loss of the base model, explicitly modulating the loss depending on the accuracy of the bias-only model. We also extend these methods to be robust against multiple sources of bias by training multiple bias-only models.

Our approaches are simple and highly effective. They require training only a simple model on top of the base model. They are model agnostic and general enough to be applicable for addressing common biases seen in many datasets in different domains.

We evaluate our models on challenging benchmarks in textual entailment and fact verification, including HANS (Heuristic Analysis for NLI Systems) (McCoy et al., 2019b), hard NLI sets (Gururangan et al., 2018) of Stanford Natural Language Inference (SNLI) (Bowman et al., 2015) and MultiNLI (MNLI) (Williams et al., 2018), and FEVER Symmetric test set (Schuster et al., 2019). The selected datasets are highly challenging and have been carefully designed to be unbiased to allow proper evaluation of the out-of-domain performance of the models. We additionally construct hard MNLI datasets from MNLI development sets to facilitate the out-of-domain evaluation on this dataset.[3] We show that including our strategies on training baseline models, including BERT (Devlin et al., 2019), provides a substantial gain on out-of-domain performance in all the experiments.

In summary, we make the following contributions: 1) Proposing two debiasing strategies to train neural models robust to dataset bias. 2) An empirical evaluation of the methods on two large-scale NLI datasets and a fact verification benchmark; obtaining a substantial gain on their challenging out-of-domain data, including 7.4 points on HANS, 4.8 points on SNLI hard set, and 9.8 points on FEVER symmetric test set, setting a new state-of-the-art. 3) Proposing debiasing strategies capable of combating multiple sources of bias. 4) Evaluating the transfer performance of the debiased models on 12 NLI datasets and demonstrating improved transfer to other NLI benchmarks.

2.2 Related Work

To address dataset biases, researchers have proposed to augment datasets by balancing the existing cues (Schuster et al., 2019) or to create an adversarial dataset (Jia and Liang, 2017). However, collecting new datasets, especially at a large scale, is costly, and thus remains an unsatisfactory solution. It is, therefore, crucial to develop strategies to allow models to be trained on the existing biased datasets.

Schuster et al. (2019) propose to first compute the n-grams in the dataset's claims that are the most associated with each fact-verification label. They then solve an optimization problem to assign a balancing weight to each training sample to alleviate the biases. In contrast, we propose several end-to-end debiasing strategies. Additionally, Belinkov et al. (2019a) propose adversarial techniques to remove from the NLI sentence encoder the features that allow a hypothesis-only model to succeed. However, we believe that in general, the features used by the hypothesis-only model can include some information necessary to perform the NLI task, and removing such information from the sentence representation can hurt the performance of the full model. Their approach consequently degrades the

[3] Removing the need to submit to an online evaluation system for MNLI hard test sets.

performance on the hard SNLI set, which is expected to be less biased. In contrast, we propose to train a bias-only model to use its predictions to dynamically adapt the classification loss to reduce the importance of the most biased examples.

Concurrently to our work, Clark et al. (2019b) and He et al. (2019) have also proposed to use the product of experts (PoE) models for avoiding biases. They train their models in two stages, first training a bias-only model and then using it to train a robust model. In contrast, our methods are trained in an end-to-end manner, which is convenient in practice. We additionally show that our proposed Debiased Focal Loss model is an effective method to reduce biases, sometimes superior to PoE. We have evaluated on new domains of NLI hard sets and fact verification. Moreover, we have included an analysis showing that our debiased models indeed have lower correlations with the bias-only models, and have extended our methods to guard against multiple bias patterns simultaneously. We furthermore study transfer performance to other NLI datasets.

2.3 Reducing Biases

Problem formulation We consider a general multi-class classification problem. Given a dataset $\mathcal{D} = \{\boldsymbol{x_i}, y_i\}_{i=1}^{N}$ consisting of the input data $\boldsymbol{x_i} \in \mathcal{X}$, and labels $y_i \in \mathcal{Y}$, the goal of the base model is to learn a mapping f_M parameterized by θ_M that computes the predictions over the label space given the input data, shown as $f_M : \mathcal{X} \rightarrow \mathbb{R}^{|\mathcal{Y}|}$. Our goal is to optimize θ_M parameters such that we build a model that is more resistant to benchmark dataset biases, to improve its robustness to domain changes where the biases typically observed in the training data do not exist in the evaluation dataset.

The key idea of our approach, depicted in Figure 2.1, is first to identify the dataset biases that the base model is susceptible to relying on, and define a bias-only model to capture them. We then propose two strategies to incorporate this bias-only knowledge into the training of the base model to make it robust against the biases. After training, we remove the bias-only model and use the predictions of the base model.

2.3.1 Bias-only Branch

We assume that we do not have access to any data from the out-of-domain dataset, so we need to know a priori about the possible types of shortcuts we would like the base model to avoid relying on. Once these patterns are identified, we train a bias-only model designed to capture the identified shortcuts that only uses *biased features*. For instance, a hypothesis-only model in the large-scale NLI datasets can correctly classify the majority of samples using annotation artifacts (Poliak et al., 2018; Gururangan et al., 2018). Motivated by this work, our bias-only model for NLI only uses hypothesis sentences. Note that the bias-only model can, in general, have any form, and is not limited to models using only a part of the input data. For instance, on the HANS dataset, our bias-only model makes use of syntactic heuristics and similarity features (see §2.4.3).

Let $\boldsymbol{x_i^b} \in \mathcal{X}^b$ be *biased features* of $\boldsymbol{x_i}$ that are predictive of y_i. We then formalize this bias-only model

as a mapping $f_B : \mathcal{X}^b \rightarrow \mathbb{R}^{|\mathcal{Y}|}$, parameterized by θ_B and trained using cross-entropy (CE) loss $\mathcal{L}_B$:

$$\mathcal{L}_B(\theta_B) = -\frac{1}{N}\sum_{i=1}^{N}\log(\sigma(f_B^{y_i}(\boldsymbol{x_i^b};\theta_B))), \tag{2.1}$$

where $f_B^j(\boldsymbol{x_i^b},\theta_B)$ is the jth element of $f_B(.)$, and $\sigma(u^j) = e^{u^j}/\sum_{k=1}^{|\mathcal{Y}|}e^{u^k}$ is the softmax function.

2.3.2 Proposed Debiasing Strategies

We propose two strategies to incorporate the bias-only f_B knowledge into the training of the base model f_M. In our strategies, the predictions of the bias-only model are combined with either the predictions of the base model or its error, to down-weight the loss for the examples that the bias-only model can predict correctly. We then update parameters of the base model θ_M based on this modified loss $\mathcal{L}_C$. Our learning strategies are end-to-end. Therefore, to prevent the base model from learning the biases, the bias-only loss $\mathcal{L}_B$ is not back-propagated to any shared parameters of the base model, such as a shared sentence encoder.

Method 1: Product of Experts

Our first approach is based on the *product of experts* (PoE) method (Hinton, 2002). Here, we use this method to combine the bias-only and base model's predictions by computing the element-wise product $\odot$ between their predictions as $\sigma(f_B(\boldsymbol{x_i^b})) \odot \sigma(f_M(\boldsymbol{x_i}))$. We compute this combination in the logarithmic space, making it appropriate for the normalized exponential below:

$$f_C(\boldsymbol{x_i},\boldsymbol{x_i^b}) = \log(\sigma(f_B(\boldsymbol{x_i^b}))) + \log(\sigma(f_M(\boldsymbol{x_i}))),$$

The key intuition behind this model is to combine the probability distributions of the bias-only and the base model to allow them to make predictions based on different characteristics of the input; the bias-only branch covers prediction based on biases, and the base model focuses on learning the actual task. Then the base model parameters θ_M are trained using the cross-entropy loss $\mathcal{L}_C$ of the combined classifier f_C:

$$\mathcal{L}_C(\theta_M;\theta_B) = -\frac{1}{N}\sum_{i=1}^{N}\log(\sigma(f_C^{y_i}(\boldsymbol{x_i},\boldsymbol{x_i^b}))). \tag{2.2}$$

When updating the base model parameters using this loss, the predictions of the bias-only model decrease the updates for examples that it can accurately predict.

Justification Probability of label y_i for the example $\boldsymbol{x_i}$ in the PoE model is computed as:

$$\sigma(f_C^{y_i}(\boldsymbol{x_i},\boldsymbol{x_i^b})) = \frac{\sigma(f_B^{y_i}(\boldsymbol{x_i^b}))\sigma(f_M^{y_i}(\boldsymbol{x_i}))}{\sum_{k=1}^{|\mathcal{Y}|}\sigma(f_B^{k}(\boldsymbol{x_i^b}))\sigma(f_M^{k}(\boldsymbol{x_i}))}$$

Then the gradient of cross-entropy loss of the combined classifier (2.2) w.r.t θ_M is (Hinton, 2002):

$$\nabla_{\theta_M}\mathcal{L}_C(\theta_M;\theta_B)=-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{|\mathcal{Y}|}\left[\Big(\delta_{y_ik}-\sigma(f_C^k(\boldsymbol{x_i},\boldsymbol{x_i^b}))\Big)\nabla_{\theta_M}\log(\sigma(f_M^k(\boldsymbol{x_i})))\right],$$

where δ_{y_ik} is 1 when $k{=}y_i$ and 0 otherwise. Generally, the closer the ensemble's prediction $\sigma(f_C^k(.))$ is to the target δ_{y_ik}, the more the gradient is decreased through the modulating term, which only happens when the bias-only and base models are both capturing biases.

In the extreme case, when the bias-only model correctly classifies the sample, $\sigma(f_C^{y_i}(\boldsymbol{x_i},\boldsymbol{x_i^b}))=1$ and therefore $\nabla_{\theta_M}\mathcal{L}_C(\theta_M;\theta_B)=0$, the biased examples are ignored during training. Conversely, when the example is fully unbiased, the bias-only classifier predicts the uniform distribution over all labels $\sigma(f_B^k(\boldsymbol{x_i^b}))=\frac{1}{|\mathcal{Y}|}$ for $k\in\mathcal{Y}$, therefore $\sigma(f_C^{y_i}(\boldsymbol{x_i},\boldsymbol{x_i^b}))=\sigma(f_M^{y_i}(\boldsymbol{x_i}))$ and the gradient of ensemble classifier remains the same as the CE loss.

Method 2: Debiased Focal Loss

Focal loss was originally proposed in Lin et al. (2017) to improve a single classifier by down-weighting the well-classified points. We propose a novel variant of this loss that leverages the bias-only branch's predictions to reduce the relative importance of the most biased examples and allows the model to focus on learning the *hard* examples. We define *Debiased Focal Loss* (DFL) as:

$$\mathcal{L}_C(\theta_M;\theta_B)=-\frac{1}{N}\sum_{i=1}^{N}\Big(1-\sigma(f_B^{y_i}(\boldsymbol{x_i^b}))\Big)^{\gamma}\log(\sigma(f_M^{y_i}(\boldsymbol{x_i}))) \tag{2.3}$$

where γ is the focusing parameter, which impacts the down-weighting rate. When γ is set to 0, DFL is equivalent to the cross-entropy loss. For $\gamma>0$, as the value of γ is increased, the effect of down-weighting is increased. We set $\gamma{=}2$ through all experiments, which works well in practice, and avoid fine-tuning it further. We note the properties of this loss: (1) When the example $\boldsymbol{x_i}$ is unbiased, and the bias-only branch does not do well, $\sigma(f_B^{y_i}(\boldsymbol{x_i^b}))$ is small, therefore the scaling factor is close to 1, and the loss remains unaffected. (2) As the sample is more biased and $\sigma(f_B^{y_i}(\boldsymbol{x_i^b}))$ is closer to 1, the modulating factor approaches 0 and the loss for the most biased examples is down-weighted.

2.3.3 RUBi baseline (Cadene et al., 2019)

We compare our models to RUBi (Cadene et al., 2019), a recently proposed model to alleviate unimodal biases learned by Visual Question Answering (VQA) models. Cadene et al. (2019)'s study is limited to VQA datasets. We, however, evaluate the effectiveness of their formulation on multiple challenging NLU benchmarks. RUBi consists in first applying a sigmoid function ϕ to the bias-only model's predictions to obtain a mask containing an importance weight between 0 and 1 for each label. It then

computes the element-wise product between the obtained mask and the base model's predictions:

$$f_C(\boldsymbol{x_i},\boldsymbol{x_i^b}) = f_M(\boldsymbol{x_i}) \odot \phi(f_B(\boldsymbol{x_i^b})),$$

The main intuition is to dynamically adjust the predictions of the base model to prevent it from leveraging the shortcuts. Then the parameters of the base model θ_M are updated by back-propagating the cross-entropy loss $\mathcal{L}_C$ of the combined classifier.

2.3.4 Joint Debiasing Strategies

Neural models can, in practice, be prone to multiple types of biases in the datasets. We, therefore, propose methods for combining several bias-only models. To avoid learning relations between biased features, we do not consider training a classifier on top of their concatenation.

Instead, let $\{\boldsymbol{x_i^{b_j}}\}_{j=1}^K$ be different sets of *biased features* of $\boldsymbol{x_i}$ that are predictive of y_i, and let f_{B_j} be an individual bias-only model capturing $\boldsymbol{x_i^{b_j}}$. Next, we extend our debiasing strategies to handle multiple bias patterns.

Method 1: Joint Product of Experts We extend our proposed PoE model to multiple bias-only models by computing the element-wise product between the predictions of bias-only models and the base model as: $\sigma(f_{B_1}(\boldsymbol{x_i^{b_1}})) \odot \cdots \odot \sigma(f_{B_K}(\boldsymbol{x_i^{b_K}})) \odot \sigma(f_M(\boldsymbol{x_i}))$, computed in the logarithmic space:

$$f_C(\boldsymbol{x_i},\{\boldsymbol{x_i^{b_j}}\}_{j=1}^K) = \sum_{j=1}^{K} \log(\sigma(f_{B_j}(\boldsymbol{x_i^{b_j}}))) + \log(\sigma(f_M(\boldsymbol{x_i}))).$$

Then the base model parameters θ_M are trained using the cross-entropy loss of the combined classifier f_C.

Method 2: Joint Debiased Focal Loss To extend DFL to handle multiple bias patterns, we first compute the element-wise average of the predictions of the multiple bias-only models: $f_B(\{\boldsymbol{x_i^{b_j}}\}_{j=1}^K) = \frac{1}{K}\sum_{j=1}^K f_{B_j}(\boldsymbol{x_i^{b_j}})$, and then compute the DFL (2.3) using the computed joint bias-only model.

2.4 Evaluation on Unbiased Datasets

We provide experiments on a fact verification (FEVER) and two large-scale NLI datasets (SNLI and MNLI). We evaluate the models' performance on recently-proposed challenging unbiased evaluation sets. We use the BERT (Devlin et al., 2019) implementation of Wolf et al. (2020) as our main baseline, known to work well for these tasks. In all the experiments, we use the default hyperparameters of the baselines.

Loss	Dev	Test	Δ
CE	85.99	56.49	
RUBi	86.23	57.60	+1.1
Schuster et al. (2019)	84.6	**61.6**	**+5.1**
DFL	83.07	64.02	+7.5
PoE	86.46	**66.25**	**+9.8**

Table 2.1: Results on FEVER development and symmetric test set. **Δ** are absolute differences with CE loss.

2.4.1 Fact Verification

Dataset The FEVER dataset contains claim-evidence pairs generated from Wikipedia. Schuster et al. (2019) collected a new evaluation set for the FEVER dataset to avoid the idiosyncrasies observed in the claims of this benchmark. They made the original claim-evidence pairs of the FEVER evaluation dataset symmetric, by augmenting them and making each claim and evidence appear with each label. Therefore, by balancing the artifacts, relying on statistical cues in claims to classify samples is equivalent to a random guess. The collected dataset is challenging, and the performance of the models relying on biases evaluated on this dataset drops significantly.

Base models We consider BERT as the base model, which works the best on this dataset (Schuster et al., 2019), and predicts the relations based on the concatenation of the claim and the evidence with a delimiter token (see Appendix 2.8).

Bias-only model The bias-only model predicts the labels using only claims as input.

Results Table 2.1 shows the results. Our proposed debiasing methods, PoE and DFL, are highly effective, boosting the performance of the baseline by 9.8 and 7.5 points respectively, significantly surpassing the prior work of Schuster et al. (2019).

2.4.2 Natural Language Inference

Datasets We evaluate on hard datasets of SNLI and MNLI (Gururangan et al., 2018), which are the splits of these datasets where a hypothesis-only model cannot correctly predict the labels. Gururangan et al. (2018) show that the success of the recent textual entailment models is attributed to the *biased* examples, and the performance of these models is substantially lower on the *hard* sets.

Base models We consider BERT and InferSent (Conneau et al., 2017) as our base models. We choose InferSent to be able to compare with the prior work of Belinkov et al. (2019b).

Loss	BERT			InferSent		
	Test	**Hard**	**Δ**	**Test**	**Hard**	**Δ**
CE	90.53	80.53		84.24	68.91	
RUBi	90.69	**80.62**	**+0.1**	83.93	**69.64**	**+0.7**
AdvCls*	—	—	—	83.56	66.27	-2.6
AdvDat*	—	—	—	78.30	55.60	-13.3
DFL	89.57	**83.01**	**+2.5**	73.54	73.05	+4.1
PoE	90.11	82.15	+1.6	80.35	**73.69**	**+4.8**

t]

Table 2.2: Results on the SNLI test, hard set, and differences with CE loss. *: results from Belinkov et al. (2019b).

Bias-only model The bias-only model predicts the labels using the hypothesis (Appendix 2.9).

Results on SNLI Table 2.2 shows the SNLI results. With InferSent, DFL and PoE result in 4.1 and 4.8 points gain. With BERT, DFL and PoE improve the results by 2.5 and 1.6 absolute points. Compared to the prior work of Belinkov et al. (2019b) (AdvCls), our PoE model obtains a 7.4 points gain, setting a new state-of-the-art.

Results on MNLI We construct hard sets from the validation sets of MNLI Matched and Mismatched (MNLI-M). Following Gururangan et al. (2018), we train a `fastText` classifier (Joulin et al., 2017) that predicts the labels using only the hypothesis and consider the subset on which it fails as hard examples.

We report the results on MNLI mismatched sets in Table 2.3 (see Appendix 2.9 for similar results on MNLI matched). With BERT, DFL and PoE obtain 1.4 and 1.7 points gain on the hard development set, while with InferSent, they improve the results by 2.5 and 2.6 points. To comply with limited access to the MNLI submission system, we evaluate only the best result of the baselines and our models on the test sets. Our PoE model improves the performance on the hard test set by 1.1 points while retaining in-domain accuracy.

2.4.3 Syntactic Bias in NLI

Dataset McCoy et al. (2019b) show that NLI models trained on MNLI can adopt superficial syntactic heuristics. They introduce HANS, consisting of several examples on which the syntactic heuristics fail.

Base model We use BERT as our base model and train it on the MNLI dataset.

Loss	BERT			InferSent		
	MNLI	Hard	Δ	MNLI	Hard	Δ
Development set results						
CE	84.53	77.55		69.99	56.53	
RUBi	85.17	78.63	+1.1	70.53	**58.08**	**+1.5**
DFL	84.85	78.92	+1.4	61.12	59.05	+2.5
PoE	84.85	**79.23**	**+1.7**	65.85	**59.14**	**+2.6**
Test set results						
CE	83.51	75.75		—	—	—
PoE	83.47	**76.83**	**+1.1**	—	—	—

Table 2.3: Results on MNLI mismatched benchmark and MNLI mismatched hard set. Δ are absolute differences with CE loss.

Loss	MNLI	HANS	Δ
CE	84.51	61.88 ±1.9	
RUBi	84.53	61.76±2.7	-0.1
Reweight ❣	83.54	**69.19**	**+7.3**
Learned-Mixin ❣	84.29	64.00	+2.1
Learned-Mixin+H ❣ ✤	83.97	**66.15**	**+4.3**
PoE	84.19	66.31±0.6	+4.4
DFL	83.95	**69.26** ±0.2	**+7.4**
DFL✤	82.76	**71.95**±1.4	**+10.1**

Table 2.4: Results on MNLI Matched dev set and HANS. ❣: results from Clark et al. (2019b). ✤: perform hyper-parameter tuning. Δ are differences with CE loss.

Bias-only model We consider the following features for the bias-only model. The first four features are based on the syntactic heuristics proposed in McCoy et al. (2019b): 1) Whether all words in the hypothesis are included in the premise; 2) If the hypothesis is the contiguous subsequence of the premise; 3) If the hypothesis is a subtree in the premise's parse tree; 4) The number of tokens shared between premise and hypothesis normalized by the number of tokens in the premise. We additionally include some similarity features: 5) The cosine similarity between premise and hypothesis's pooled token representations from BERT followed by min, mean, and max-pooling. We consider the same weight for contradiction and neutral labels in the bias-only loss to allow the model to recognize entailment from not-entailment. During the evaluation, we map the neutral and contradiction labels to not-entailment.

Results McCoy et al. (2019a) observe large variability in the linguistic generalization of neural models. We, therefore, report the averaged results across 4 runs with the standard deviation in Table 2.4. PoE and DFL obtain 4.4 and 7.4 points gain (see Appendix 2.10 for accuracy on individual heuristics of HANS).

Loss	MNLI	Hard	Δ	HANS	Δ
CE	84.53	77.55		61.88±1.9	
PoE ♣	84.85	**79.23**	**+1.7**	60.43	-1.5
DFL♣	84.85	78.92	+1.4	60.63	-1.2
PoE ♥	84.55	77.90±0.3	+0.4	66.31±0.6	+4.4
DFL♥	84.30	77.66±0.6	+0.1	**69.26**±0.2	**+7.4**
PoE-Joint	84.39	**78.61**±0.1	**+1.1**	68.04±1.2	+6.2
DFL-Joint	84.49	78.36±0.4	+0.8	**69.10**±0.7	**+7.2**

Table 2.5: Results on MNLI mismatched dev set, MNLI mismatched hard set, and HANS when training independently to debias against either hypothesis artifacts (♣) or syntactic biases (♥), compared with jointly training to debias against both bias types. **Δ**: differences with baseline CE loss.

We compare our results with the concurrent work of Clark et al., who propose a PoE model similar to ours, which gets similar results. The main difference is that our models are trained end-to-end, which is convenient in practice, while Clark et al.'s method requires two steps, first training a bias-only model and then using this pre-trained model to train a robust model. The Reweight baseline in Clark et al. is a special case of our DFL with $\gamma = 1$ and performs similarly to our DFL method (using default $\gamma = 2$). Their Learned-Mixin+H method requires hyperparameter tuning. Since the assumption is not having access to any out-of-domain test data, and there is no available dev set for HANS, it is challenging to perform hyper-parameter tuning. Clark et al. follow prior work (Grand and Belinkov, 2019; Ramakrishnan et al., 2018) and perform model selection on the test set.

To provide a fair comparison, we consequently also tuned γ in DFL by sweeping over $\{0.5,1,2,3,4\}$. DFL✤ is the selected model, with $\gamma = 3$. With this hyperparameter tuning, DFL is even more effective, and our best result performs 2.8 points better than Clark et al. (2019b).

2.4.4 Jointly Debiasing Multiple Bias Patterns

To evaluate combating multiple bias patterns, we jointly debias a base model on the hypothesis artifacts and syntactic biases.

Base model We use BERT as our base model and train it on the MNLI dataset.

Bias-only models We use the hypothesis-only and syntactic bias-only models as in §2.4.2 and §2.4.3.

Results Table 2.5 shows the results. Models trained to be robust to hypothesis biases (♣) do not generalize to HANS. On the other hand, models trained to be robust on HANS (♥) use a powerful bias-only model resulting in a slight improvement on MNLI mismatched hard dev set. We expect a

slight degradation when debiasing for both biases since models need to select samples accommodating both debiasing needs. The jointly debiased models successfully obtain improvements on both datasets, which are close to the improvements on each dataset by the individually debiased models.

2.5 Transfer Performance

To evaluate how well the baseline and proposed models generalize to solving textual entailment in domains that do not share the same annotation biases as the large NLI training sets, we take trained NLI models and test them on several NLI datasets.

Datasets We consider a total of 12 different NLI datasets. We use the 11 datasets studied by Poliak et al. (2018). These datasets include MNLI, SNLI, SciTail (Khot et al., 2018), AddOneRTE (ADD1) (Pavlick and Callison-Burch, 2016), Johns Hopkins Ordinal Commonsense Inference (JOCI) (Zhang et al., 2017), Multiple Premise Entailment (MPE) (Lai et al., 2017), Sentences Involving Compositional Knowledge (SICK) (Marelli et al., 2014), and three datasets from White et al. (2017) which are automatically generated from existing datasets for other NLP tasks including: Semantic Proto-Roles (SPR) (Reisinger et al., 2015), Definite Pronoun Resolution (DPR) (Rahman and Ng, 2012), FrameNet Plus (FN+) (Pavlick et al., 2015), and the GLUE benchmark's diagnostic test (Wang et al., 2019c). We additionally consider the Quora Question Pairs (QQP) dataset, where the task is to determine whether two given questions are semantically matching (duplicate) or not. As in Gong et al. (2017), we interpret duplicate question pairs as an entailment relation and neutral otherwise. We use the same split ratio mentioned by Wang et al. (2017).

Since the datasets considered have different label spaces, when evaluating on each target dataset, we map the model's labels to the corresponding target dataset's space. See Appendix 2.11 for more details.

We strictly refrained from using any out-of-domain data when evaluating on the unbiased split of the same benchmark in §2.4. However, as shown by prior work (Belinkov et al., 2019a), since different NLI target datasets contain different amounts of the bias found in the large-scale NLI dataset, we need to adjust the amount of debiasing according to each target dataset. We consequently introduce a hyperparameter α for PoE to modulate the strength of the bias-only model in ensembling. We follow prior work (Belinkov et al., 2019a) and perform model selection on the dev set of each target dataset and then report results on the test set.[4] We select hyper-parameters γ, α from $\{0.4,0.6,0.8,2,3,4,5\}$.

Results Table 2.6 shows the results of the debiased models and baseline with BERT. As shown in prior work (Belinkov et al., 2019a), the MNLI datasets have very similar biases to SNLI, which the models are trained on, so we do not expect any improvement in the relative performance of our models and the baseline for MNLI and MNLI-M. On all the remaining datasets, our proposed models perform better than the baseline, showing a substantial improvement in generalization by using our debasing

[4]Since the test sets are not available for MNLI, we tune on the matched dev set and evaluate on the mismatched dev set or vice versa. For GLUE, we tune on MNLI mismatched dev set.

Data	CE	DFL	Δ	PoE	Δ
SICK	57.05	57.91	+0.9	57.28	+0.2
ADD1	87.34	88.89	+1.5	87.86	+0.5
DPR	49.50	50.68	+1.2	50.14	+0.6
SPR	59.85	61.41	+1.6	62.45	+2.6
FN+	53.16	54.77	+1.6	53.51	+0.4
JOCI	50.06	51.13	+1.1	50.85	+0.8
MPE	69.50	70.2	+0.7	70.1	+0.6
SCITAIL	67.64	69.33	+1.7	71.40	+3.8
GLUE	54.08	54.80	+0.7	54.71	+0.6
QQP	67.78	69.28	+1.5	68.61	+0.8
MNLI	74.40	73.58	-0.8	73.61	-0.8
MNLI-M	73.98	74.0	0.0	73.49	-0.5

Table 2.6: Accuracy results of models with BERT transferring to new target datasets. All models are trained on SNLI and tested on the target datasets. **Δ** are absolute differences between our methods and the CE loss baseline.

techniques. We additionally compare with Belinkov et al. (2019a) in Appendix 2.11 and show that our methods substantially surpass their results.

2.6 Discussion

Analysis of Debiased Focal Loss As expected, improving the out-of-domain performance could come at the expense of decreased in-domain performance since the removed biases are useful for performing the in-domain task. This happens especially for DFL, in which there is a trade-off between in-domain and out-of-domain performance that depends on the parameter γ, and when the baseline model is not very powerful like InferSent. To understand the impact of γ in DFL, we train an InferSent model using DFL for different values of γ on the SNLI dataset and evaluate its performance on SNLI test and SNLI hard sets. As illustrated in Figure 2.2, increasing γ increases debiasing and thus hurts in-domain accuracy on SNLI, but out-of-domain accuracy on the SNLI hard set is increased within a wide range of values (see a similar plot for BERT in Appendix 2.12).

Correlation Analysis In contrast to Belinkov et al. (2019a), who encourage only the encoder to not capture the unwanted biases, our learning strategies influence the parameters of the full model to reduce the reliance on unwanted patterns more effectively. To test this assumption, in Figure 2.3, we report the correlation between the element-wise loss of the debiased models and the loss of a bias-only model on the considered datasets.

The results show that compared to the baselines, our debiasing methods, DFL and PoE, reduce the correlation to the bias-only model, confirming that our models are effective at reducing biases. Interestingly, on MNLI, PoE has less correlation with the bias-only model than DFL and also has better performance

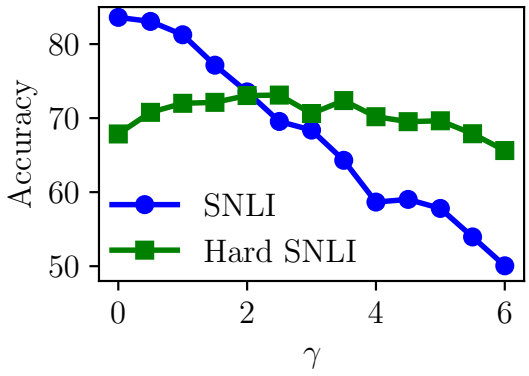

Figure 2.2: Accuracy of InferSent model trained with DFL, on the SNLI test and SNLI hard sets for different γ.

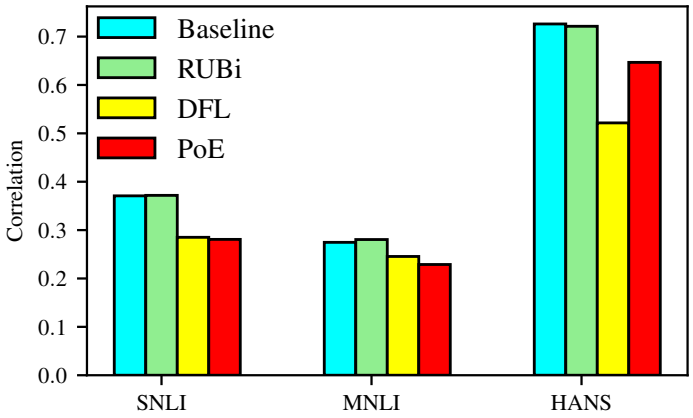

Figure 2.3: Pearson correlation between the element-wise cross-entropy loss of the debiasing models and the bias-only model trained on each dataset.

on the unbiased split of this dataset. On the other hand, on the HANS dataset, DFL loss is less correlated with the bias-only model than PoE and also obtains higher performance on the HANS dataset.

2.7 Conclusion

In this chapter, we propose two novel techniques, product-of-experts and debiased focal loss, to reduce biases learned by neural models, which are applicable whenever one can specify the biases in the form of one or more bias-only models. The bias-only models are designed to leverage biases and shortcuts in the datasets. Our debiasing strategies then work by adjusting the cross-entropy loss based on the performance of these bias-only models, to focus learning on the hard examples and down-weight the importance of the biased examples. Additionally, we extend our methods to combat multiple bias patterns simultaneously. Our proposed debiasing techniques are model agnostic, simple, and highly effective. Extensive experiments show that our methods substantially improve the model robustness

to domain-shift, including 9.8 points gain on FEVER symmetric test set, 7.4 on HANS dataset, and 4.8 points on SNLI hard set. Furthermore, we show that our debiasing techniques result in better generalization to other NLI datasets. Future work may include developing debiasing strategies that do not require prior knowledge of bias patterns and can automatically identify them.

Appendix

2.8 Fact Verification

Base model We fine-tune all models using BERT for 3 epochs and use the default parameters and default learning rate of $2e{-}5$.

Bias-only model Our bias-only classifier is a shallow nonlinear classifier with 768, 384, 192 hidden units with Tanh nonlinearity.

2.9 Natural Language Inference

Base model InferSent uses a separate BiLSTM encoder to learn sentence representations for premise and hypothesis. It then combines these embeddings following Mou et al. (2016) and feeds them to the default nonlinear classifier. With InferSent we train all models for 20 epochs as default without using early-stopping. We use the default hyper-parameters and following Wang et al. (2019c), we set the BiLSTM dimension to 512. We use the default nonlinear classifier with 512 and 512 hidden neurons with Tanh nonlinearity. With BERT, we finetune all models for 3 epochs.

Bias-only model For debiasing models using BERT, we use the same shallow nonlinear classifier explained in Appendix 2.8, and for the ones using InferSent, we use a shallow linear classifier with 512 and 512 hidden units.

Results Table 2.7 shows results on the MNLI matched development and hard test sets.

2.10 Syntactic Bias in NLI

Base model We finetune all models for 3 epochs.

Bias-only model We use a nonlinear classifier with 6 and 6 hidden units with Tanh nonlinearity.

Loss	BERT			InferSent		
	MNLI	Hard	Δ	MNLI	Hard	Δ
Development set results						
CE	84.41	**76.56**		69.97	**57.03**	
RUBi	84.48	77.13	+0.6	70.51	57.97	+0.9
DFL	83.72	77.37	+0.8	60.78	57.88	+0.9
PoE	84.58	**78.02**	**+1.5**	66.02	**59.37**	**+2.3**
Test set results						
None	84.11	75.88		—	—	—
PoE	84.11	**76.81**	**+0.9**	—	—	—

Table 2.7: Results on the MNLI matched benchmark and MNLI matched hard set. **Δ** are absolute differences with CE loss.

Loss	HANS		
	Constituent	Lexical	Subsequence
gold label: Entailment			
CE	98.98±0.6	96.41±0.8	99.72±0.1
RUBi	99.22±0.3	95.59±0.8	99.50±0.3
DFL	90.90±4.3	84.78±5.0	94.33±4.9
PoE	97.24±1.9	92.16±0.9	98.58±0.5
gold label: Non-entailment			
CE	20.12±5.8	48.86±5.7	7.18±0.7
RUBi	21.89±7.0	46.82±12.5	7.58±2.3
DFL	50.20±9.2	71.06±3.1	24.28±4.4
PoE	36.08±5.1	59.18±8.0	14.63±3.0

Table 2.8: Accuracy for each label (entailment or non-entailment) on individual heuristics of HANS.

Results Table 2.8 shows the performance for each label (entailment and non_entailment) on individual heuristics of the HANS dataset.

2.11 Transfer Performance

Mapping We train all models on SNLI and evaluate their performance on other target datasets. SNLI contains three labels, contradiction, neutral, and entailment. Some of the datasets we consider contain only two labels. In the case of labels *entailed* and *not-entailed*, as in DPR, we map contradiction and neutral to the not-entailed class. In the case of labels *entailment* and *neutral*, as in SciTail, we map

Data	CE	DFL	Δ%	PoE	Δ%	M1	Δ%	M2	Δ%
SICK	54.09	55.00	1.68	55.79	3.14	49.77	-7.99	49.77	-7.99
ADD1	75.19	78.29	4.12	77.00	2.41	67.44	-10.31	67.44	-10.31
DPR	49.95	50.59	1.28	49.95	0.00	50.87	1.84	50.87	1.84
SPR	41.31	47.95	16.07	50.50	22.25	51.51	24.69	51.51	24.69
FN+	48.65	49.58	1.91	49.35	1.44	53.23	9.41	53.23	9.41
JOCI	46.47	46.48	0.02	47.53	2.28	44.83	-3.53	44.83	-3.53
MPE	60.60	60.70	0.17	61.80	1.98	56.40	-6.93	56.40	-6.93
SCITAIL	64.25	65.19	1.46	63.17	-1.68	56.40	-12.22	56.40	-12.22
GLUE	48.73	46.83	-3.90	49.09	0.74	43.93	-9.85	43.93	-9.85
QQP	61.80	66.24	7.18	66.36	7.38	62.46	1.07	62.46	1.07
MNLI	56.99	56.70	-0.51	56.59	-0.70	51.72	-9.25	51.72	-9.25
MNLI-M	57.01	57.75	1.30	57.84	1.46	53.99	-5.30	53.99	-5.30
Average	—	—	2.57	—	3.39	—	-2.36	—	-2.36

Table 2.9: Accuracy results of models with InferSent transferring to new target datasets. All models are trained on SNLI and tested on the target datasets. M1 and M2 are our re-implementation of Belinkov et al. (2019a). **Δ** are relative differences in percentage with respect to CE loss.

contradiction to neutral.

Comparison with Belinkov et al. (2019a) We modified the implementations of Belinkov et al. (2019a) and corrected some implementation issues in the InferSent baseline Conneau et al. (2017). Compared to the original InferSent implementation, the main differences in our implementation include: (a) We incorporated the fixes suggested for the bugs in the implementation of mean/max-pooling over BiLSTM in the InferSent baseline[5] (b). We additionally observed that the aggregation of losses over each batch was computed with the average instead of the intended summation and we corrected it.[6] (c) We followed the implementation of InferSent and we removed out-of-vocabulary (OOV) words from the sentence representation, while Belinkov et al. keep them by introducing an OOV token. We additionally observed during the pre-processing of some of the target datasets in the implementation of Belinkov et al., some of the samples are not considered due to the preprocessing issues. We fix the pre-processing issues and evaluate our models and our reimplementations of Belinkov et al. (2019a) on the same corpora. We set the BiLSTM dimension to 512 across all models. Note that Belinkov et al. use BiLSTM dimension of 2048, and due to the mentioned differences in implementations and datasets, the results reported in Belinkov et al. (2019a) are not comparable. However, we still on average surpass their reported results substantially. Our reimplementations and scripts to reproduce the results are publicly available in `https://github.com/rabeehk/robust-nli-fixed`.

As used in prior work to adjust the learning-rate of the bias-only and baseline models (Belinkov et al.,

[5] `https://github.com/facebookresearch/InferSent/issues/51`

[6] The same observation is reported in `https://github.com/facebookresearch/InferSent/pull/107`.

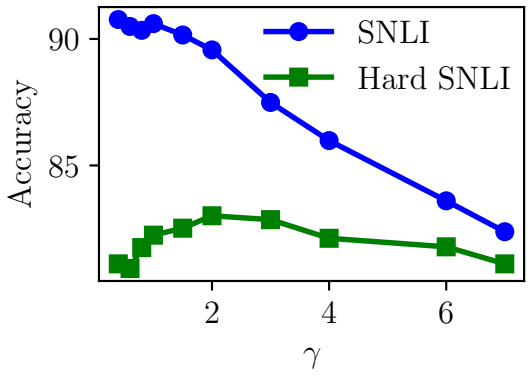

Figure 2.4: Accuracy of the BERT model trained with DFL, on SNLI and SNLI hard sets for different γ.

2019a), we introduce a hyperparameter β for the bias-only model to modulate the loss of the bias-only model in ensembling. We sweep hyper-parameters γ, α over $\{0.02,0.05,0.1,0.6,2.0,4.0,5.0\}$ and β over $\{0.05,0.2,0.4,0.8,1.0\}$. Table 2.9 shows the results of our debiasing models (DFL, PoE), our re-implementations of proposed methods in Belinkov et al. (2019a) (M1, M2), and the baseline with InferSent (CE). The DFL model outperforms the baseline in 10 out of 12 datasets, while the PoE model outperforms the baseline in 9 datasets and does equally well on the DPR dataset. As shown in prior work (Belinkov et al., 2019a), the MNLI dataset has very similar biases to SNLI, which the models are trained on, so we do not expect any improvement in the relative performance of our models and the baseline for MNLI dataset. Interestingly, our methods obtain improvement on MNLI-M, in which the test data differs from training distribution. Our proposed debiasing methods, PoE and DFL, are highly effective, boosting the relative generalization performance of the baseline by 3.39% and 2.57% respectively, significantly surpassing the prior work of Belinkov et al. (2019a). Compared to M1 and M2, our methods outperform them on 9 datasets, while they do better on two datasets of SPR and FN+, and slightly better on the DPR dataset. However, note that DPR is a very small dataset and all models perform close to random-chance on this dataset.

2.12 Analysis of Debiased Focal Loss

Figure 2.4 shows the impact of γ on BERT trained with DFL.

3 Variational Information Bottleneck for Effective Low-Resource Fine-Tuning

While large-scale pretrained language models have obtained impressive results when fine-tuned on a wide variety of tasks, they still often suffer from overfitting in low-resource scenarios. Since such models are general-purpose feature extractors, many of these features are inevitably irrelevant for a given target task. In this chapter, we propose to use Variational Information Bottleneck (VIB) to suppress irrelevant features when fine-tuning on low-resource target tasks, and show that our method successfully reduces overfitting. Moreover, we show that our VIB model finds sentence representations that are more robust to biases in natural language inference datasets, and thereby obtains better generalization to out-of-domain datasets. Evaluation on seven low-resource datasets in different tasks shows that our method significantly improves transfer learning in low-resource scenarios, surpassing prior work. Moreover, it improves generalization on 13 out of 15 out-of-domain natural language inference benchmarks.[1]

3.1 Introduction

Transfer learning has emerged as the de facto standard technique in natural language processing (NLP), where large-scale language models are pretrained on an immense amount of text to learn a general-purpose representation, which is then transferred to the target domain with fine-tuning on target task data. This method has exhibited state-of-the-art results on a wide range of NLP benchmarks (Devlin et al., 2019; Liu et al., 2019b; Radford et al., 2019). However, such pretrained models have a huge number of parameters, potentially making fine-tuning susceptible to overfitting.

In particular, the task-universal nature of large-scale pretrained sentence representations means that much of the information in these representations is irrelevant to a given target task. If the amount of target task data is small, it can be hard for fine-tuning to distinguish relevant from irrelevant information, leading to overfitting on statistically spurious correlations between the irrelevant information and target labels. Learning low-resource tasks is an important topic in NLP (Cherry et al., 2019) because annotating more data can be very costly and time-consuming, and because in several tasks access to data is limited.

[1] Our code is publicly available in `https://github.com/rabeehk/vibert`.

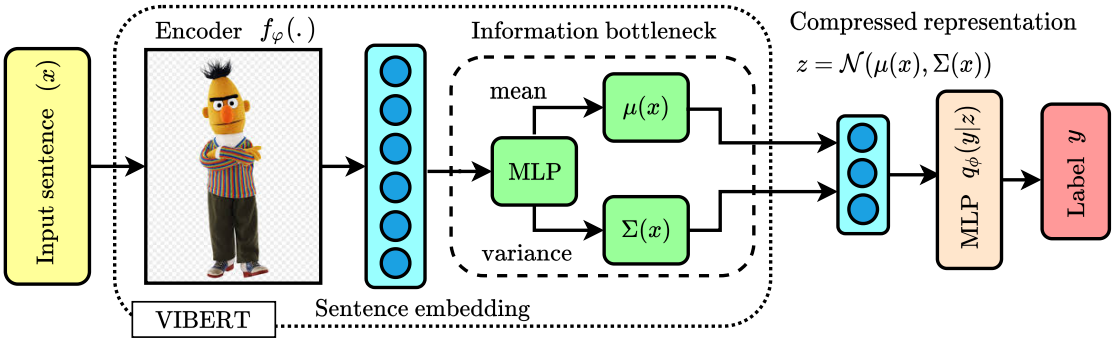

Figure 3.1: VIBERT compresses the encoder's sentence representation $f_\varphi(x)$ into representation z with mean $\mu(x)$ and eliminates irrelevant and redundant information through the Gaussian noise with variance $\Sigma(x)$.

In this chapter, we propose to use the Information Bottleneck (IB) principle (Tishby et al., 1999) to address this problem of overfitting. More specifically, we propose a fine-tuning method that uses Variational Information Bottleneck (VIB; Alemi et al. 2017) to improve transfer learning in low-resource scenarios.

VIB addresses the problem of overfitting by adding a regularization term to the training loss that directly suppresses irrelevant information. As illustrated in Figure 3.1, the VIB component maps the sentence embedding from the pretrained model to a latent representation z, which is the only input to the task-specific classifier. The information that is represented in z is chosen based on the IB principle, namely that all the information about the input that is represented in z should be necessary for the task. In particular, VIB directly tries to remove the irrelevant information, making it easier for the task classifier to avoid overfitting when trained on a small amount of data. We find that in low-resource scenarios, using VIB to suppress irrelevant features in pretrained sentence representations substantially improves accuracy on the target task.

Removing unnecessary information from the sentence representation also implies removing redundant information. VIB tries to find the most concise representation which can still solve the task, so even if a feature is useful alone, it may be removed if it isn't useful when added to other features because it is redundant. We hypothesize that this provides a useful inductive bias for some tasks, resulting in better generalization to out-of-domain data. In particular, it has recently been demonstrated that annotation biases and artifacts in several natural language understanding benchmarks (Kaushik and Lipton, 2018; Gururangan et al., 2018; Poliak et al., 2018; Schuster et al., 2019) allow models to exploit superficial shortcuts during training to perform surprisingly well without learning the underlying task. However, models that rely on such superficial features do not generalize well to out-of-domain datasets, which do not share the same shortcuts (Belinkov et al., 2019a). We investigate whether using VIB to suppress redundant features in pretrained sentence embeddings has the effect of removing these superficial shortcuts and keeping the deep semantic features that are truly useful for learning the underlying task. We find that using VIB does reduce the model's dependence on shortcut features and substantially improves generalization to out-of-domain datasets.

We evaluate the effectiveness of our method on fine-tuning BERT (Devlin et al., 2019), which we call the VIBERT model (Variational Information Bottleneck for Effective Low-Resource Fine-Tuning). On seven different datasets for text classification, natural language inference, similarity, and paraphrase tasks, VIBERT shows greater robustness to overfitting than conventional fine-tuning and other regularization techniques, improving accuracies on low-resource datasets. Moreover, on NLI datasets, VIBERT shows robustness to dataset biases, obtaining substantially better generalization to out-of-domain NLI datasets. Further analysis demonstrates that VIB regularization results in less biased representations. Our approach is highly effective and simple to implement, involving a small additional MLP classifier on top of the sentence embeddings. It is model agnostic and end-to-end trainable.

In summary, we make the following contributions: 1) Proposing VIB for low-resource fine-tuning of large pretrained language models. 2) Showing empirically that VIB reduces overfitting, resulting in substantially improved accuracies on seven low-resource benchmark datasets against conventional fine-tuning and prior regularization techniques. 3) Showing empirically that training with VIB is more robust to dataset biases in NLI, resulting in significantly improved generalization to out-of-domain NLI datasets.

3.2 Fine-tuning in Low-resource Settings

The standard fine-tuning paradigm starts with a large-scale pretrained model such as BERT, adds a task-specific output component which uses the pretrained model's sentence representation, and trains this model end-to-end on the task data, fine-tuning the parameters of the pretrained model. As depicted in Figure 3.1, we propose to add a VIB component that controls the flow of information from the representations of the pretrained model to the output component. The goal is to address overfitting in resource-limited scenarios by removing irrelevant and redundant information from the pretrained representation.

Problem Formulation We consider a general multi-class classification problem with a low-resource dataset $\mathcal{D}=\{x_i,y_i\}_{i=1}^{N}$ consisting of inputs $x_i \in \mathcal{X}$, and labels $y_i \in \mathcal{Y}$. We assume we are also given a large-scale pretrained encoder $f_\varphi(.)$ parameterized by φ that computes sentence embeddings for the input x_i. Our goal is to fine-tune $f_\varphi(.)$ on $\mathcal{D}$ to maximize generalization.

Information Bottleneck To specifically optimize for the removal of irrelevant and redundant information from the input representations, we adopt the Information Bottleneck principle. The objective of IB is to find a maximally compressed representation Z of the input representation X (compression loss) that maximally preserves information about the output Y (prediction loss),[2] by minimizing:

[2]In this work, Z, X, and Y are random variables, and z, x and y are instances of these random variables.

$$\mathcal{L}_{\text{IB}} = \underbrace{\beta I(X, Z)}_{\text{Compression Loss}} - \underbrace{I(Z, Y)}_{\text{Prediction Loss}} , \tag{3.1}$$

where $\beta \geq 0$ controls the balance between compression and prediction, and $I(.,.)$ is the mutual information.

Variational Information Bottleneck Alemi et al. (2017) derive an efficient variational estimate of (3.1):

$$\mathcal{L}_{\text{VIB}} = \beta \mathop{\mathbb{E}}_{x}[\text{KL}[p_\theta(z|x), r(z)]] + \mathop{\mathbb{E}}_{z \sim p_\theta(z|x)}[-\log q_\phi(y|z)], \tag{3.2}$$

where $q_\phi(y|z)$ is a parametric approximation of $p(y|z)$, $r(z)$ is an estimate of the prior probability $p(z)$ of z, and $p_\theta(z|x)$ is an estimate of the posterior probability of z. During training, the compressed sentence representation z is sampled from the distribution $p_\theta(z|x)$, meaning that a specific pattern of noise is added to the input of the output classifier $q_\phi(y|z)$. Increasing this noise decreases the information conveyed by z. In this way, the VIB module can block the output classifier $q_\phi(y|z)$ from learning to use specific information. At test time, the expected value of z is used for predicting labels with $q_\phi(y|z)$. We refer to the dimensionality of z as K, which specifies the bottleneck size. Note that there is an interaction between decreasing K and increasing the compression by increasing β (Shamir et al., 2010; Harremoës and Tishby, 2007). K and β are hyper-parameters (Alemi et al., 2017).

We consider parametric Gaussian distributions for prior $r(z)$ and $p_\theta(z|x)$ to allow an analytic computation for their Kullback-Leibler divergence,[3] namely $r(z) = \mathcal{N}(z|\mu_0, \Sigma_0)$ and $p_\theta(z|x) = \mathcal{N}(z|\mu(x), \Sigma(x))$, where μ and μ_0 are $K-$dimensional mean vectors, and Σ and Σ_0 are diagonal covariance matrices. We use the reparameterization trick (Kingma and Welling, 2013) to estimate the gradients, namely $z = \mu(x) + \Sigma(x) \odot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$. To compute the compressed sentence representations $p_\theta(z|x)$, as shown in Figure 3.1, we first feed sentence embeddings $f_\varphi(x)$ through a shallow MLP. It is then followed by two linear layers, each with K hidden units to compute $\mu(x)$ and $\Sigma(x)$ (after a softplus transform to ensure non-negativity). We also use another linear layer to approximate $q_\phi(y|z)$.

3.3 Experiments

Datasets We evaluate the performance on seven different benchmarks for multiple tasks, in particular text classification, natural language inference, similarity, and paraphrase detection. For NLI, we experiment with two well-known NLI benchmarks, namely SNLI (Bowman et al., 2015) and MNLI (Williams et al., 2018). For text classification, we evaluate on two sentiment analysis datasets, namely IMDB (Maas et al., 2011) and Yelp2013 (YELP) (Zhang et al., 2015). We additionally evaluate

[3] $\text{KL}(\mathcal{N}(\mu_0, \Sigma_0) \| \mathcal{N}(\mu_1, \Sigma_1)) = \frac{1}{2}(\text{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - K + \log(\frac{\det(\Sigma_1)}{\det(\Sigma_0)}))$.

on three low-resource datasets in the GLUE benchmark (Wang et al., 2019c):[4] paraphrase detection using MRPC (Dolan and Brockett, 2005), semantic textual similarity using STS-B (Cer et al., 2017), and textual entailment using RTE (Dagan et al., 2005). For the GLUE benchmark, SNLI, and Yelp, we evaluate on the standard validation and test splits. For MNLI, since the test sets are not available, we tune on the matched dev set and evaluate on the mismatched dev set (MNLI-M) or vice versa. See Appendix 3.7 for datasets statistics and Appendix 3.8 for hyper-parameters of all methods.

Base Model We use the $\text{BERT}_{\text{Base}}$ (12 layers, 110M parameters) and $\text{BERT}_{\text{Large}}$ (24 layers, 340M parameters) uncased (Devlin et al., 2019) implementation of Wolf et al. (2020) as our base models,[5] known to work well for these tasks. We use the default hyper-parameters of BERT, i.e., we use a sequence length of 128, with batch size 32. We use the stable variant of the Adam optimizer (Zhang et al., 2021b; Mosbach et al., 2021) with the default learning rate of $2\text{e}{-5}$ through all experiments. We do not use warm-up or weight decay.

Baselines We compare against prior regularization techniques, including previous state-of-the-art, Mixout:

- **Dropout** (Srivastava et al., 2014), a widely used stochastic regularization techniques used in multiple large-scale language models (Devlin et al., 2019; Yang et al., 2019; Vaswani et al., 2017) to mitigate overfitting. Following Devlin et al. (2019), we apply dropout on all layers of BERT.

- **Mixout** (Lee et al., 2019) is a stochastic regularization technique inspired by Dropout with the goal of preventing catastrophic forgetting during fine-tuning. Mixout regularizes the learning to minimize the deviation of a fine-tuned model from the pretrained initialization. It replaces the model parameters with the corresponding value from the pretrained model with probability p.

- **Weight Decay (WD)** is a common regularization technique to improve generalization (Krogh and Hertz, 1992). It regularizes the large weights w by adding a penalization term $\frac{\lambda}{2}\|w\|$ to the loss, where λ is a hyperparameter specifying the strength of regularization. Chelba and Acero (2004) and Daumé III (2007) adapt WD for fine-tuning of the pretrained models, and propose to replace this regularization term with $\lambda\|w-w_0\|$, where w_0 are the weights of the pretrained models. Recently, Lee et al. (2019) demonstrated that the latter formulation of WD works better for fine-tuning of BERT than conventional WD and can improve generalization on small training sets.

[4] We did not evaluate on WNLI and CoLA due to the irregularities in these datasets and the reported instability during the fine-tuning `https://gluebenchmark.com/faq`.

[5] To have a controlled comparison, all results are computed with this PyTorch implementation, which might slightly differ from the TensorFlow variant (Devlin et al., 2019).

Table 3.1: Average results and standard deviation in parentheses over 3 runs on low-resource data in GLUE. Δ shows the absolute difference between the results of the VIBERT model with BERT.

Model	MRPC		STS-B		RTE
	Accuracy	**F1**	**Pearson**	**Spearman**	**Accuracy**
$\text{BERT}_{\text{Base}}$	87.80 (0.5)	83.20 (0.6)	84.93 (0.1)	83.53 (0.0)	67.93 (1.5)
+Dropout (Srivastava et al., 2014)	87.33 (0.2)	81.90 (0.7)	84.33 (0.9)	82.73(1.0)	65.80 (1.5)
+Mixout (Lee et al., 2019)	87.03 (0.2)	82.63 (0.3)	85.23 (0.4)	83.80(0.4)	67.70 (0.9)
+WD (Lee et al., 2019)	87.57(0.2)	82.83(0.3)	85.0(0.3)	83.6(0.2)	68.63(1.3)
$\text{VIBERT}_{\text{Base}}$	**89.23 (0.1)**	**85.23 (0.2)**	**87.63 (0.3)**	**86.50 (0.4)**	**70.53 (0.5)**
Δ	+1.43	+2.03	+2.7	+2.97	+2.6
$\text{BERT}_{\text{Large}}$	88.47 (0.7)	84.20 (1.3)	86.87 (0.2)	85.70 (0.1)	68.67 (0.8)
+Dropout (Srivastava et al., 2014)	87.77 (0.4)	82.97 (0.2)	86.47 (0.1)	85.33 (0.2)	65.77 (0.6)
+Mixout (Lee et al., 2019)	88.57 (0.7)	84.10 (1.1)	86.70 (0.2)	85.43 (0.3)	70.03 (1.0)
+WD (Lee et al., 2019)	88.97(0.5)	84.87(0.4)	86.9(0.1)	85.67(0.1)	69.27(0.9)
$\text{VIBERT}_{\text{Large}}$	**89.10 (0.4)**	**85.13 (0.6)**	**87.53 (0.8)**	**86.40 (0.9)**	**71.37 (0.8)**
Δ	+0.63	+0.93	+0.66	+0.7	+2.7

3.3.1 Results on the GLUE Benchmark

Table 3.1 shows results on the low-resource datasets in GLUE.[6] We find that a) Our VIBERT model substantially outperforms the baselines on all the datasets, demonstrating the effectiveness of the proposed method. b) Dropout decreases the performance on low-resource datasets. We conjecture that regularization techniques relying on stochasticity without considering the relevance to the output, in contrast to VIB, can make it more difficult for learning to extract relevant information from a small amount of data. Igl et al. (2019) observe similar effects in another application. c) Similar to the results of Zhang et al. (2021b), we find less pronounced benefits of the previously suggested methods than the results originally published. This can be explained by using a more stable version of Adam (Zhang et al., 2021b) suggested by the very recent work in our experiments, which decreases the added benefits of previously suggested regularization techniques on top of a stable optimizer. In contrast, our VIBERT model still substantially improves the results and surpasses the prior work in all settings for both $\text{BERT}_{\text{Base}}$ and $\text{BERT}_{\text{Large}}$ models. Due to the computational overhead of $\text{BERT}_{\text{Large}}$, for the rest of this chapter, we stick to $\text{BERT}_{\text{Base}}$.

Impact of Random Seeds Following Dodge et al. (2020), we examine the choice of random seed and evaluate the performance of VIBERT and BERT by fine-tuning them across 50 random seeds on GLUE. To comply with the limited access to the GLUE benchmark online system, we split the original validation sets into half and consider one half as the validation set and use the other half as the test set. We first perform model selection on the validation set to fix the hyper-parameters and then fine-tune the

[6]Note that the test sets are not publicly available and the prior work reports the results on the validation set of the GLUE benchmark (Lee et al., 2019; Dodge et al., 2020). We, however, report the results of their methods and ours on the original test sets by submitting to an online system.

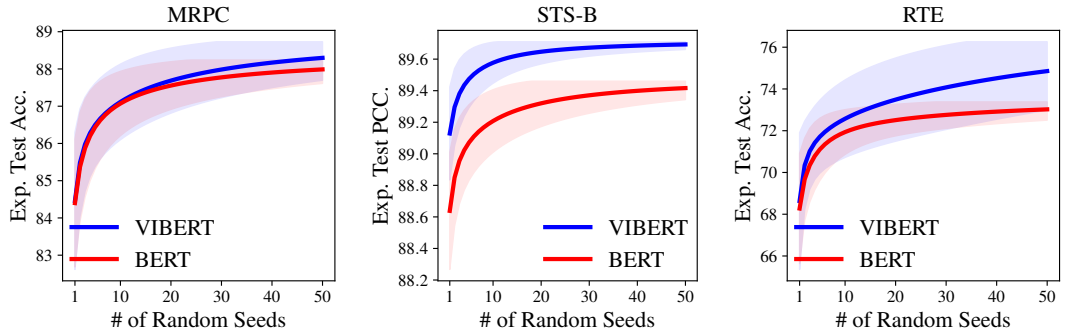

Figure 3.2: Expected test performance (solid lines) with standard deviation (shaded region) over the number of random seeds allocated for fine-tuning. Our VIBERT model consistently outperforms BERT. We report the accuracy for RTE and MRPC and the Pearson correlation coefficient for STS-B.

selected models for 50 different seeds. Figure 3.2 shows the expected test performance (Dodge et al., 2019) as the function of random trials. The results demonstrate that our VIBERT model consistently obtains better performance than BERT on all datasets. As anticipated, the expected test performance monotonically increases with more random trials (Dodge et al., 2020) till it reaches a plateau, such as after 30 trials on STS-B.

3.3.2 Varying-resource Results

To analyze the performance of our method as a function of dataset size, we use four large-resource NLI and sentiment analysis datasets, namely SNLI, MNLI, IMDB, and YELP to be able to subsample the training data with varying sizes. Table 3.2 shows the obtained results. VIBERT consistently outperforms all the baselines on low-resource scenarios, but the advantages are reduced or eliminated as we approach a medium-resource scenario. Also, the improvements are generally larger when the datasets are smaller, showing that our method successfully addresses low-resource scenarios.

3.3.3 Out-of-domain Generalization

Besides improving fine-tuning on low-resource data by removing irrelevant features, we expect VIB to improve on out-of-domain data because it removes redundant features. In particular, annotation artifacts create shortcut features, which are superficial cues correlated with a label (Gururangan et al., 2018; Poliak et al., 2018) that do not generalize well to out-of-domain datasets (Belinkov et al., 2019a). Since solving the real underlying task can be done without these superficial shortcuts, they must be redundant with the deep semantic features that are truly needed. We hypothesize that many more superficial shortcut features are needed to reach the same level of performance as a few deep semantic features. If so, then VIB should prefer to keep the concise deep features and remove the abundant superficial features, thus encouraging the classifier to rely on the deep semantic features, and therefore

Table 3.2: Test accuracies in the low-resource setting on text classification and NLI datasets under varying sizes of training data (200, 500, 800, 1000, 3000, and 6000 samples). We report the average and standard deviation in parentheses across three runs. We show the highest average result in each setting in bold. **Δ** shows the absolute difference between the results of VIBERT with BERT.

Data	Model	200	500	800	1000	3000	6000
SNLI	BERT	58.70 (1.3)	68.12 (1.5)	73.29 (0.9)	74.69 (1.1)	79.57 (0.4)	80.85 (0.4)
	+Dropout	58.95 (0.4)	69.33 (1.1)	73.22 (1.2)	74.20 (0.5)	79.48 (0.7)	**81.71** (0.6)
	+Mixout	58.52 (1.3)	68.26 (1.7)	72.81 (1.0)	74.09 (0.5)	78.7 (0.3)	80.61 (0.5)
	+WD	59.23 (1.5)	68.54 (1.9)	73.72 (1.0)	74.78 (0.8)	**79.83** (0.5)	81.32 (0.5)
	VIBERT	**61.42** (1.3)	**70.75** (0.6)	**74.71** (0.5)	**75.84** (0.1)	79.56 (0.3)	81.29 (0.4)
	Δ	+2.72	+2.63	+1.42	+1.15	-0.01	+0.44
MNLI	BERT	49.93 (1.4)	59.76 (2.0)	63.63 (1.6)	65.21 (1.4)	70.67 (0.7)	73.11 (0.9)
	+Dropout	50.74 (2.1)	59.58 (2.1)	62.82 (0.8)	65.71 (1.4)	71.11 (0.8)	72.88 (1.1)
	+Mixout	50.05 (1.8)	58.69 (2.8)	63.31 (1.7)	64.58 (1.5)	70.60 (0.8)	72.56 (0.7)
	+WD	49.92 (1.4)	60.36 (2.0)	64.41 (1.5)	65.3 (1.0)	71.47 (0.8)	72.94 (0.7)
	VIBERT	**53.58** (0.9)	**63.04** (1.1)	**64.87** (0.6)	**66.41** (1.2)	**71.86** (0.9)	**74.22** (0.3)
	Δ	+3.65	+3.28	+1.24	+1.2	+1.19	+1.11
IMDB	BERT	78.96 (1.9)	83.68 (0.2)	84.04 (0.9)	84.80 (0.0)	86.17 (0.2)	86.98 (0.4)
	+Dropout	81.19 (1.6)	83.30 (0.2)	84.52 (0.3)	85.01 (0.3)	86.20 (0.2)	**87.31** (0.2)
	+Mixout	79.17 (4.2)	83.55 (0.3)	84.37 (0.3)	84.50 (0.1)	86.15 (0.1)	86.97 (0.1)
	+WD	79.78 (2.2)	83.95 (0.2)	84.29 (0.6)	84.97 (0.2)	86.13 (0.3)	87.2 (0.1)
	VIBERT	**83.05** (0.3)	**84.46** (0.4)	**84.83** (0.4)	**85.03** (0.4)	**86.27** (0.4)	87.15 (0.3)
	Δ	+4.09	+0.78	+0.79	+0.23	+0.1	+0.17
YELP	BERT	41.60 (0.9)	44.12 (1.4)	45.67 (1.6)	46.77 (0.5)	50.14 (0.7)	51.86 (0.4)
	+Dropout	41.30 (0.3)	44.37 (0.6)	46.49 (0.8)	46.21 (1.5)	**51.09** (0.2)	**52.39** (0.5)
	+Mixout	41.52 (0.9)	43.60 (1.1)	45.65 (1.9)	46.98 (1.1)	50.68 (0.5)	51.51 (0.3)
	+WD	41.66 (0.6)	44.43 (1.2)	46.26 (1.4)	47.37 (0.6)	50.7 (0.5)	51.9 (0.6)
	VIBERT	**42.30** (0.2)	**46.65** (0.5)	**46.60** (0.1)	**48.03** (0.6)	50.37 (0.4)	51.34 (0.4)
	Δ	+0.7	+2.53	+0.93	+1.26	+0.23	-0.52

resulting in better generalization to out-of-domain data. To evaluate out-of-domain generalization, we take NLI models trained on medium-sized 6K subsampled SNLI and MNLI in §3.3.2 and evaluate their generalization on several NLI datasets.

Datasets We consider a total of 15 different NLI datasets used in Mahabadi et al. (2020), including SICK (Marelli et al., 2014), ADD1 (Pavlick and Callison-Burch, 2016), JOCI (Zhang et al., 2017), MPE (Lai et al., 2017), MNLI, SNLI, SciTail (Khot et al., 2018), and three datasets from White et al. (2017) namely DPR (Rahman and Ng, 2012), FN+ (Pavlick et al., 2015), SPR (Reisinger et al., 2015), and Quora Question Pairs (QQP) interpreted as an NLI task as by Gong et al. (2017). We use the same split used in Wang et al. (2017). We also consider SNLI hard and MNLI(-M) Hard sets (Gururangan et al., 2018), a subset of SNLI/MNLI(-M) where a hypothesis-only model cannot correctly predict the labels and the known biases are avoided. Since the target datasets have different label spaces,

Table 3.3: Test accuracy of models transferring to new target datasets. All models are trained on SNLI or MNLI and tested on the target datasets. **Δ** are absolute differences with BERT.

Data	SNLI					MNLI				
	BERT	VIBERT	Δ	WD	Δ	BERT	VIBERT	Δ	WD	Δ
SICK	48.47	54.68	+6.2	48.37	-0.1	59.16	69.17	+10.0	63.87	+4.7
ADD1	78.81	84.75	+5.9	80.62	+1.8	66.15	82.95	+16.8	67.18	+1.0
DPR	50.78	50.14	-0.6	50.41	-0.4	49.95	49.95	0.0	49.95	0.
SPR	50.21	65.68	+15.5	51.90	+1.7	59.16	65.61	+6.5	57.21	-1.9
FN+	50.78	53.44	+2.7	50.58	-0.2	46.28	49.94	+3.7	46.34	+0.1
JOCI	42.03	50.66	+8.6	43.91	+1.9	45.60	53.94	+8.3	46.49	+0.9
MPE	58.30	58.10	-0.2	58.10	-0.2	55.10	50.30	-4.8	58.2	+3.1
SCITAIL	62.32	74.84	+12.5	65.10	+2.8	72.58	75.68	+3.1	75.73	+3.2
QQP	65.19	70.67	+5.5	65.90	+0.7	67.88	70.50	+2.6	68.75	+0.9
SNLI Hard	65.72	68.35	+2.6	66.82	+1.1	56.98	60.29	+3.3	57.8	+0.8
MNLI Hard	46.31	53.17	+6.9	47.42	+1.1	59.74	61.19	+1.4	60.08	+0.3
MNLI-M Hard	46.12	52.38	+6.3	46.82	+0.7	60.55	61.03	+0.5	59.77	-0.8
SNLI	80.54	81.81	+1.3	81.26	+0.7	64.32	67.87	+3.6	65.44	+1.1
MNLI-M	60.51	64.88	+4.4	62.11	+1.6	72.42	73.06	+0.6	72.76	+0.3
MNLI	61.79	66.76	+5.0	63.42	+1.6	72.73	74.67	+1.9	72.89	+0.2
Average	—	—	+5.51	—	+0.99	—	—	+3.83	—	+0.93

during the evaluation, we map predictions to each target dataset's space (Appendix 3.9). Following prior work (Belinkov et al., 2019a; Mahabadi et al., 2020), we select hyper-parameters based on the development set of each target dataset and report the results on the test set.

Results Table 3.3 shows the results of VIBERT and BERT. We additionally include WD, the baseline that performed the best on average on SNLI and MNLI in Table 3.2. On models trained on SNLI, VIBERT improves the transfer on 13 out of 15 datasets, obtaining a substantial average improvement of 5.51 points. The amount of improvement on different datasets varies, with the largest improvement on SPR and SciTail with +15.5, and +12.5 points respectively, while WD on average obtains only 0.99 points improvement. On models trained on MNLI, VIBERT improves the transfer on 13 datasets, obtaining an average improvement of 3.83 points. The improvement varies across the datasets, with the largest on ADD1 and JOCI with 16.8 and 8.3 points respectively, substantially surpassing WD. Interestingly, VIBERT improves the results on the SNLI and MNLI(-M) hard sets, resulting in models that are more robust to known biases. These results support our claim that VIBERT motivates learning more general features, rather than redundant superficial features, leading to an improved generalization to datasets without these superficial biases. In the next section, we analyze this phenomenon more.

Table 3.4: Hypothesis-only accuracy when freezing the encoder from models trained on SNLI/MNLI in Table 3.2 and retraining a hypothesis-only classifier (BERT, VIBERT), and baseline results when the encoder is not frozen (H-only). Lower results show more successful debiasing.

Model	SNLI			MNLI		
	Train	**Dev**	**Test**	**Train**	**Dev**	**Test**
H-only	81.3	61.89	62.17	87.15	53.46	53.63
BERT	66.40	53.73	53.17	58.5	44.68	44.03
VIBERT	38.20	**36.65**	**37.10**	42.03	**36.43**	**35.75**

3.4 Analysis

Analysis of the Removed Features Elazar and Goldberg (2018) propose a challenging framework to evaluate if debiasing methods have succeeded in removing biases from the sentence representation. After debiasing, the trained encoder is frozen and the classifier is retrained to try to extract the biases. If the classifier reaches high accuracy given only bias features, then the encoder's representation has not been successfully debiased. We follow the framework of Elazar and Goldberg (2018) to analyze whether known biases in NLI data have been removed in the trained sentence representations. In particular, following Belinkov et al. (2019b), we train a classifier which only sees the representation of the hypothesis sentence and see if it can predict the class of the sentence pair, which is an established criterion to measure known biases in NLI datasets (Gururangan et al., 2018). Thus, we freeze the trained encoders from our model and the BERT baseline and retrain a hypothesis-only classifier on hypotheses from the SNLI and MNLI datasets.[7] For reference, we compare to a hypothesis-only model with a BERT encoder trained end-to-end. Table 3.4 shows the results. With the baseline (BERT), the retrained classifier is not able to recapture all the biases (H-only), but it captures much more than with our method (VIBERT). VIBERT is so successful at reducing biases that performance of the hypothesis-only classifier is close to chance (33%).

Impact of VIB on Overfitting To analyze the effect of VIB on reducing overfitting, we analyze the effect of the β parameter on training and validation error since β controls the trade-off between removing information from the sentence embedding (high β) and keeping information that is predictive of the output (low β). We fix the bottleneck size (K) based on the models selected in §3.3.1, and we train VIBERT on the GLUE benchmark for varying values of β and plot the validation and training loss in Figure 3.3.

For small values of β, where VIB has little effect, the validation loss is substantially higher than the training loss, indicating overfitting. This is because the network learns to be more deterministic ($\Sigma \approx 0$), thereby retaining too much irrelevant information. As we increase β, where VIB has an effect, we observe better generalization performance with less overfitting. As β becomes too large, both the

[7] Note that with VIBERT, the frozen encoder $p_\theta(z|x)$ outputs a distribution, and the hypothesis-only classifier is trained on samples from this distribution.

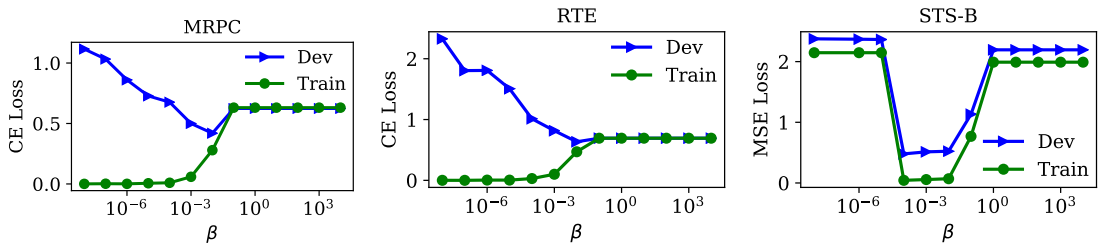

Figure 3.3: Validation and training losses of VIBERT for varying β and a fixed bottleneck size on GLUE.

Table 3.5: Performance evaluation for all methods. $\Delta\%$ are relative differences with BERT.

Model	**Memory**	**Δ%**	**#Parameters**	**Δ%**	**Time**	**Δ%**
BERT	290.91 GB	—	109.48 M	—	4.50 min	—
+Mixout	407.65 GB	40.13 %	109.48 M	0%	5.15 min	14.44%
+WD	331.78 GB	14.05%	109.48 M	0%	4.91 min	9.11%
+Dropout	290.91 GB	0%	109.48 M	0%	4.68 min	4%
VIBERT	292.57 GB	0.57 %	110.83 M	1.22%	4.67 min	3.77%

training and validation losses shoot up because the amount of preserved information is insufficient to differentiate between the classes. This pattern is observable in the MRPC and RTE datasets, with a similar pattern in the STS-B dataset.

Efficiency Evaluation Table 3.5 presents the efficiency evaluation in terms of memory, number of parameters, and time for all the methods measured on RTE. Our approach has several attractive properties. First, while our method is slightly larger in terms of parameters compared to the other standard regularization approaches due to an additional MLP layer (Figure 3.1), the difference is still marginal, and for $\text{BERT}_{\text{Base}}$ model with 109.48M trainable parameters, that is less than 1.22% more parameters. Second, our approach presents a much better memory usage with low-overhead, close to Dropout, while WD and especially Mixout cause substantial memory overhead. In dealing with large-scale transformer models like BERT, efficient memory usage is of paramount importance. Third, in terms of training time, our method is similar to Dropout and much faster than the other two baselines. Relative to BERT, VIBERT increases the training time by 3.77%, while WD and Mixout cause the substantial training overhead of 9.11% and 14.44%. Note that our method and other baselines require hyper-parameter tuning.

Ablation Study As an ablation, Table 3.6 shows results for our model without the compression loss (VIBERT ($\beta = 0$)), in which case there is no incentive to introduce noise, and the VIB layer reduces to deterministic dimensionality reduction with an MLP. We optimize the dimensionality of the MLP layer (K) as a hyper-parameter for both methods. This ablation does reduce performance on all considered datasets, demonstrating the added benefit of the compression loss of VIBERT.

Table 3.6: Average ablation results over 3 runs with std in parentheses on GLUE. BERT and VIBERT's results are from Table 3.1.

Model	MRPC		STS-B		RTE
	Accuracy	**F1**	**Pearson**	**Spearman**	**Accuracy**
BERT	87.80 (0.5)	83.20 (0.6)	84.93 (0.1)	83.53 (0.0)	67.93 (1.5)
VIBERT (β=0)	88.57 (0.6)	84.27 (0.7)	87.10 (0.4)	86.00 (0.5)	69.63 (1.3)
VIBERT	**89.23** (0.1)	**85.23** (0.2)	**87.63** (0.3)	**86.50** (0.4)	**70.53** (0.5)

3.5 Related Work

Low-resource Setting Recently, developing methods for low-resource NLP has gained attention (Cherry et al., 2019). Prior work has investigated improving on low-resource datasets by injecting large unlabeled in-domain data and pretraining a unigram document model using a variational autoencoder and use its internal representations as features for downstream tasks (Gururangan et al., 2019). Other approaches propose injecting a million-scale previously collected phrasal paraphrase relations (Arase and Tsujii, 2019) and data augmentation for translation task (Fadaee et al., 2017). Due to relying on the additional source and in-domain corpus, such techniques are not directly comparable to our model.

Information Bottleneck IB has recently been adopted in NLP in applications such as parsing (Li and Eisner, 2019), and summarization (West et al., 2019). Voita et al. (2019) use the mutual information to study how token representations evolve across layers of a Transformer model (Vaswani et al., 2017). Our method – to the best of our knowledge – is the first attempt to study VIB as a regularization technique to improve the fine-tuning of large-scale language models on low-resource scenarios.

Regularization Techniques for Fine-tuning Language models In addition to references given throughout, Phang et al. (2018) proposed to perform an extra data-rich intermediate supervised task pretraining followed by fine-tuning on the target task. They showed that their method leads to improved fine-tuning performance on the GLUE benchmark. However, their method requires pretraining with a large intermediate task. In contrast, our goal is to use only the provided low-resource target datasets.

3.6 Conclusion and Future Directions

In this chapter, we propose VIBERT, an effective model to reduce overfitting when fine-tuning large-scale pretrained language models on low-resource datasets. By leveraging a VIB objective, VIBERT finds the simplest sentence embedding, predictive of the target labels, while removing task-irrelevant

and redundant information. Our approach is model agnostic, simple to implement, and highly effective. Extensive experiments and analyses show that our method substantially improves transfer performance in low-resource scenarios. We demonstrate our obtained sentence embeddings are robust to biases and our model results in a substantially better generalization to out-of-domain NLI datasets. Future work includes exploring incorporating VIB on multiple layers of pretrained language models and using it to jointly learn relevant features and relevant layers.

Appendix

3.7 Experimental Details

Datasets Statistics Table 3.7 shows the statistics of the datasets used in our experiments.

Computing Infrastructure We run all experiments on one GTX1080Ti GPU with 11 GB of RAM.

VIBERT Architecture The MLP module used to compute the compressed sentence representations (Figure 3.1) is a shallow MLP with 768, $\frac{2304+K}{4}$, $\frac{768+K}{2}$ hidden units with a ReLU non-linearity, where K is the bottleneck size. Following Alemi et al. (2017), we average over 5 posterior samples, i.e., we compute $p(y|x) = \frac{1}{5}\Sigma_{i=1}^{5} q_\phi(y|z_i)$, where $z^i \sim p_\theta(z|x)$. Similar to Bowman et al. (2016), we use a linear annealing schedule for β and set it as $\min(1, \text{epoch} \times \beta_0)$ in each epoch, where β_0 is the initial value.

3.8 Hyper-parameters

The GLUE Benchmark Experiment Results on GLUE benchmark are reported in Table 3.1. We fine-tune all the models for 6 epochs to allow them to converge. We use early stopping for all models by choosing the model performing the best on the validation set with the evaluation criterion of average F1 and accuracy for MRPC, accuracy for RTE, and average Pearson and Spearman correlations for STS-B. For VIBERT, we sweep β over $\{10^{-4}, 10^{-5}, 10^{-6}\}$ and K over $\{144, 192, 288, 384\}$. For dropout, we use dropping probabilities of $\{0.25, 0.45, 0.65, 0.85\}$. For Mixout, we consider mixout probability of $\{0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9\}$. For WD, we consider weight decay of $\{10^{-6},10^{-5},10^{-4},10^{-3},10^{-2},10^{-1},1\}$.

Varying-resource Experiment Results on varying sizes of training data are reported in Table 3.2. We fine-tune all models for 25 epochs to allow them to converge. We use early stopping for all models based on the performance on the validation set. We also perform hyper-parameter tuning on the validation set. Since we consider datasets of a different number of training samples, we need to account for a suitable range of bottleneck size and we sweep K over $\{12,18,24,36,48,72,96,144,192,288,384\}$ and β over $\{10^{-4},10^{-5}\}$. For dropout, we consider dropping probabilities of $\{0.25,0.45,0.65,0.85\}$.

Table 3.7: Datasets used in our experiments.

Dataset	#Labels	Train	Val.	Test
Single-Sentence Tasks				
IMDB	2	20K	5K	25K
YELP	5	62.5K	7.8K	8.7K
Inference Tasks				
SNLI	3	550K	10K	10K
MNLI	3	393K	9.8K	9.8K
RTE	2	2.5K	0.08K	3K
Similarity and Paraphrase Tasks				
MRPC	2	3.7K	0.4K	1.7K
STS-B	1 (Similarity score)	5.8K	1.5K	1.4K

For Mixout, we consider mixout probability of $\{0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9\}$. For WD, we consider weight decay of $\{10^{-6},10^{-5},10^{-4},10^{-3},10^{-2},10^{-1},1\}$.

Ablation Experiment Ablation results are shown in Table 3.6. For VIBERT (β=0), we sweep K over the same range of values as VIBERT, i.e., $\{144,192,288,384\}$

3.9 Mapping

We train all models on SNLI or MNLI datasets and evaluate their performance on other target datasets. The SNLI and MNLI datasets contain three labels of contradiction, neutral, and entailment. However, some of the considered target datasets have only two labels, such as DPR or SciTail. When the target dataset has two labels of *entailed* and *not-entailed*, as in DPR, we consider the predicted contradiction and neutral labels as the not-entailed label. In the case the target dataset has two labels of *entailment* and *neutral*, as in SciTail, we consider the predicted contradiction label as neutral.

4 Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks

State-of-the-art parameter-efficient fine-tuning methods rely on introducing adapter modules between the layers of a pretrained language model. However, such modules are trained separately for each task and thus do not enable sharing information across tasks. In this chapter, we show that we can learn adapter parameters for all layers and tasks by generating them using shared hypernetworks, which condition on task, adapter position, and layer id in a transformer model. This parameter-efficient multi-task learning framework allows us to achieve the best of both worlds by sharing knowledge across tasks via hypernetworks while enabling the model to adapt to each individual task through task-specific adapters. Experiments on the well-known GLUE benchmark show improved performance in multi-task learning while adding only 0.29% parameters per task. We additionally demonstrate substantial performance improvements in few-shot domain generalization across a variety of tasks.[1]

4.1 Introduction

Transfer learning from pretrained large-scale language models yields state-of-the-art results in a variety of tasks (Devlin et al., 2019; Radford et al., 2018; Liu et al., 2019b). As a highly expressive and abstract framework, Raffel et al. (2020) explored the landscape of transfer learning by converting text-based natural language processing (NLP) problems into a sequence-to-sequence format to train a unified model on several tasks simultaneously. Multi-task learning with pretrained language models (Ruder, 2017) is appealing for multiple reasons: 1) Training individual models per task results in higher computational costs, which hinders deployment and maintenance. These costs are substantially reduced by training a single model. 2) Fine-tuning the model across multiple tasks allows sharing information between the different tasks and positive transfer to other related tasks. Specifically, when target datasets have limited training data, multi-task learning improves the performance compared to individually trained models (Liu et al., 2019a; Ratner et al., 2018). However, multi-task fine-tuning

[1] Our code is publicly available in `https://github.com/rabeehk/hyperformer`.

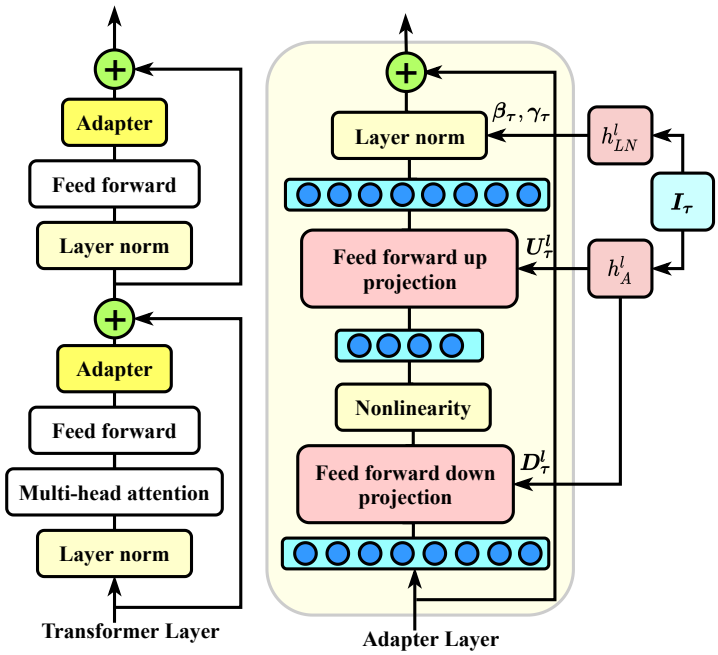

Figure 4.1: Left: Adapter integration in the T5 model. Right: Our HYPERFORMER adapter architecture. Following Houlsby et al. (2019), we include adapter modules after the two feed-forward layers. The Adapter hypernetwork h^l_A produces the weights ($\boldsymbol{U^l_\tau}$ and $\boldsymbol{D^l_\tau}$) for task-specific adapter modules conditioned on an input task embedding $\boldsymbol{I_\tau}$. Similarly, the layer normalization hypernetwork h^l_{LN} generates the conditional layer normalization parameters ($\boldsymbol{\beta_\tau}$ and $\boldsymbol{\gamma_\tau}$). During training, we only update layer normalizations in T5, hypernetworks, and task embeddings. The compact HYPERFORMER++ shares the same hypernetworks across all layers and tasks and computes the task embedding based on task, layer id, and position of the adapter module (§4.2.4).

can result in models underperforming on high-resource tasks due to constrained capacity (Arivazhagan et al., 2019; McCann et al., 2018). An additional issue with multi-task fine-tuning is the potential for *task interference* or *negative transfer*, where achieving good performance on one task can hinder performance on another (Wang et al., 2019d).

As an alternative to fine-tuning (Howard and Ruder, 2018), adapter layers (Houlsby et al., 2019) insert a small number of additional parameters per task into the model. During fine-tuning, only the adapter modules, layer normalizations, and parameters of the final classification layer are updated, while the original pretrained model parameters remain frozen. Such task-specific adapters eliminate negative task interference by encapsulating task-specific information (Pfeiffer et al., 2020). However, so far there has not been an effective and parameter-efficient way to share information across multiple adapters to enable positive transfer to low-resource and related tasks.

To address this problem and to enable sharing information across tasks while reaping the benefits of adapter layers, as depicted in Figure 4.1, we propose HYPERFORMER++, which employs a compact hypernetwork (Ha et al., 2017; Oswald et al., 2020) shared across tasks and layers. The hypernetwork learns to generate task and layer-specific adapter parameters, conditioned on task and layer id em-

beddings. The hypernetwork is jointly learned between all tasks and is thus able to share information across them, while negative interference is minimized by generating separate adapter layers for each task. For each new task, our model only requires learning an additional task embedding, reducing the number of trained parameters.

We use the encoder-decoder T5 model (Raffel et al., 2020) as the underlying model for our experiments and evaluate on the standard GLUE benchmark (Wang et al., 2019c). We achieve strong gains over both the T5_{BASE} model as well as adapters (Houlsby et al., 2019). To our knowledge, this is the first time that adapters have been successfully integrated into a state-of-the-art encoder-decoder model beyond machine translation (Bapna and Firat, 2019), demonstrating that our method effectively balances sharing information across tasks while minimizing negative transfer.

In summary, we make the following contributions: (1) We propose a parameter-efficient method for multi-task fine-tuning based on hypernetworks and adapter layers. (2) We demonstrate that our method scales more efficiently than prior work. (3) We provide empirical results on GLUE demonstrating the effectiveness of the proposed method on multi-task learning. (4) We perform extensive few-shot domain transfer experiments, which reveal that the captured shared knowledge can positively transfer to unseen in-domain tasks.

4.2 HYPERFORMER

In this section, we present our HYPERFORMER model, which integrates **hyper**network-based adapter layers into a multi-task trans**former** model. In §4.2.4, we introduce a parameter-efficient variant of this model, called HYPERFORMER++.

Problem formulation We consider a general multi-task learning problem, where we are given the data from a set of tasks $\{\mathcal{D}_\tau\}_{\tau=1}^{T}$, where T is the total number of tasks and $\mathcal{D}_\tau = \{(\boldsymbol{x}_\tau^i, y_\tau^i)\}_{i=1}^{N_\tau}$ shows the training data for τ-th task with N_τ samples. We assume we are also given a large-scale pretrained language model $f_{\boldsymbol{\theta}}(.)$ parameterized by $\boldsymbol{\theta}$ that computes the output for input $\boldsymbol{x}_\tau^i$. Standard multi-task fine-tuning minimizes the following loss on the training set:

$$\mathcal{L}(\boldsymbol{\theta}, \{\mathcal{D}_\tau\}_{\tau=1}^{T}) = \sum_{\tau=1}^{T} \sum_{(\boldsymbol{x}_\tau^i, y_\tau^i) \in \mathcal{D}_\tau} w_\tau l\Big(f_{\boldsymbol{\theta}}(\boldsymbol{x}_\tau^i), y_\tau^i\Big), \tag{4.1}$$

where l is typically the cross-entropy loss, and w_τ shows the sampling weight for τ-th task. Our goal is to finetune the pretrained model in a multi-task learning setup efficiently, while allowing sharing information across tasks and at the same time, enabling the model to adapt to each individual task.

The key idea of our approach, depicted in Figure 4.1, is to learn a parametric task embedding $\{\boldsymbol{I}_\tau\}_{\tau=1}^{T}$ for each task, and then feed these task embeddings to hypernetworks parameterized by $\boldsymbol{\nu}$ that generate the task-specific adapter layers (Houlsby et al., 2019). We insert adapter modules within the layers of a pretrained model, making the final model of $\mathcal{X}_{\boldsymbol{\nu}}(\boldsymbol{x}_\tau^i, \boldsymbol{\theta}, \boldsymbol{I}_\tau)$ parameterized by $\boldsymbol{\nu}$ that computes the output

for input $\boldsymbol{x}_{\boldsymbol{\tau}}^{\boldsymbol{i}}$. During training, we only train hypernetwork parameters $\boldsymbol{\nu}$, task embeddings $\{\boldsymbol{I}_{\boldsymbol{\tau}}\}_{\tau=1}^{T}$, and layer normalizations in $f_\theta(.)$, while the rest of the pretrained model parameters $\boldsymbol{\theta}$ are fixed:

$$\mathcal{L}(\boldsymbol{\nu},\{\boldsymbol{I}_{\boldsymbol{\tau}}\}_{i=1}^{T},\{\mathcal{D}_\tau\}_{\tau=1}^{T})=\sum_{\tau=1}^{T}\sum_{(\boldsymbol{x}_{\boldsymbol{\tau}}^{\boldsymbol{i}},y_\tau^i)\in\mathcal{D}_\tau} w_\tau l\Big(\mathcal{X}_{\boldsymbol{\nu}}(\boldsymbol{x}_{\boldsymbol{\tau}}^{\boldsymbol{i}},\boldsymbol{\theta},\boldsymbol{I}_{\boldsymbol{\tau}}),y_\tau^i\Big), \tag{4.2}$$

The hypernetworks capture the shared information across tasks in a multi-task learning model enabling positive transfer between related domains and transferable tasks, while adapters are reducing negative interference, encapsulating task-specific information.

Base model All of our models are built on top of the state-of-the-art T5 transformer model (Raffel et al., 2020). This model frames text-based language tasks as sequence-to-sequence problems. T5 consists of an encoder-decoder Transformer (Vaswani et al., 2017) with minor modifications (Raffel et al., 2020). The model is trained simultaneously on multiple tasks, obtaining state-of-the-art performance across a diverse set of tasks. We use the T5 framework as it enables training a universal model that interfaces with many language tasks. Our model has three main components: 1) task conditional adapter layers; 2) task conditional layer normalizations; and 3) hypernetworks that generate task-specific parameters. We next describe these components.

4.2.1 Task Conditional Adapter Layers

Prior work has shown that fine-tuning all parameters of the model can result in a sub-optimal solution, particularly for resource-limited datasets (Peters et al., 2019). As an alternative to fine-tuning all the model's parameters, prior work (Houlsby et al., 2019; Rebuffi et al., 2018; Stickland and Murray, 2019) inserted small modules called *adapter layers* within layers of a pretrained model, as shown in Figure 4.1. Adapters introduce no change to the structure or parameters of the original model.

In this chapter, we propose conditional adapter modules, in which we generate the adapters weights based on input task embeddings using shared hypernetworks (Ha et al., 2017), which capture information across tasks that can be used to positively transfer to other relevant tasks.

Each layer of a transformer model consists of an attention block and a feed-forward block, each followed by a skip connection. Following Houlsby et al. (2019), as depicted in Figure 4.1, we introduce a conditional adapter layer after each block before the skip connection. The conditional adapter layer A_τ^l for layer l consists of a down-projection, $\boldsymbol{D}_{\boldsymbol{\tau}}^{\boldsymbol{l}}\in\mathbb{R}^{h\times d}$, GeLU non-linearity (Hendrycks and Gimpel, 2016), and up-projection $\boldsymbol{U}_{\boldsymbol{\tau}}^{\boldsymbol{l}}\in\mathbb{R}^{d\times h}$, where h is the input dimension, and d is the bottleneck dimension for the adapter layer, mathematically defined as:

$$A_\tau^l(\boldsymbol{x})=LN_\tau^l\Big(\boldsymbol{U}_{\boldsymbol{\tau}}^{\boldsymbol{l}}(\text{GeLU}(\boldsymbol{D}_{\boldsymbol{\tau}}^{\boldsymbol{l}}(\boldsymbol{x})))\Big)+\boldsymbol{x}, \tag{4.3}$$

where $\boldsymbol{x}$ is the input hidden state and LN_τ^l is the conditional layer norm defined in the next section. We generate adapter weights ($\boldsymbol{U}_{\boldsymbol{\tau}}^{\boldsymbol{l}}$, $\boldsymbol{D}_{\boldsymbol{\tau}}^{\boldsymbol{l}}$) through a hypernetwork described in §4.2.3.

4.2.2 Task Conditional Layer Normalization

Conventional layer normalization (Ba et al., 2016) is defined as:

$$LN^l_\tau(\boldsymbol{x^i_\tau}) = \boldsymbol{\gamma^l_\tau} \odot \frac{\boldsymbol{x^i_\tau} - \boldsymbol{\mu_\tau}}{\boldsymbol{\sigma_\tau}} + \boldsymbol{\beta^l_\tau}, \tag{4.4}$$

where $\odot$ is the element-wise multiplication between two vectors, and $\boldsymbol{\gamma^l_\tau}$ and $\boldsymbol{\beta^l_\tau}$ are learnable parameters with the same dimension as $\boldsymbol{x^i_\tau}$. Values of $\boldsymbol{\mu_\tau}$ and $\boldsymbol{\sigma_\tau}$ show the mean and standard deviation of training data for the τ-th task.

To allow the layer normalization inside adapters to adapt to each task, inspired by Perez et al. (2018); De Vries et al. (2017), we generate $\boldsymbol{\gamma^l_\tau}$, $\boldsymbol{\beta^l_\tau}$ via a hypernetwork as a function of task embeddings (§4.2.3).

4.2.3 Task Conditioned Hypernetworks

In order to have a model that can share information while being able to adapt to each individual task, we generate the parameters of task conditional adapter layers and layer normalization using hypernetworks. A hypernetwork is a network that generates the weights of another network (Ha et al., 2017).

The hypernetworks capture the shared information, while the generated task conditional adapters and layer normalization allow the model to adapt to each individual task to reduce negative task interference.

Learned task embedding We first compute a task embedding $\boldsymbol{I_\tau} \in \mathbb{R}^t$ for each individual task using a task projector network $h_I(.)$, which is a multi-layer perceptron consisting of two feed-forward layers and a ReLU non-linearity:

$$\boldsymbol{I_\tau} = h_I(\boldsymbol{z_\tau}), \tag{4.5}$$

where $\boldsymbol{z_\tau} \in \mathbb{R}^{t'}$ can be a learnable parameter or any pretrained task features (Vu et al., 2020), and the task projector network $h_I(.)$ learns a suitable compressed task embedding from input task features. In this chapter, we consider a parametric $\boldsymbol{z_\tau}$ to allow end-to-end training which is convenient in practice.[2]

Removing task prefixes The T5 model prepends task-specific prefixes to the input sequence for conditioning. For instance, when training on CoLA (Warstadt et al., 2019), *cola sentence:* is prepended to each sample. Instead, we remove task prefixes and use task embeddings for conditioning.

Task conditioned hypernetworks We consider simple linear layers as hypernetworks that are functions of input task embeddings $\boldsymbol{I_\tau}$. We introduce these hypernetworks in each layer of the transformer. We define hypernetwork $h^l_A(.)$ that generates task conditional adapter weights ($\boldsymbol{U^l_\tau}$, $\boldsymbol{D^l_\tau}$):

$$(\boldsymbol{U^l_\tau}, \boldsymbol{D^l_\tau}) := h^l_A(\boldsymbol{I_\tau}) = \left(\boldsymbol{W^{U^l}}, \boldsymbol{W^{D^l}}\right)\boldsymbol{I_\tau}, \tag{4.6}$$

[2] We ran some pilot experiments with pretrained task embeddings (Vu et al., 2020), but did not observe extra benefits.

where $\boldsymbol{W}^{\boldsymbol{U}^{\boldsymbol{l}}} \in \mathbb{R}^{(d\times h)\times t}$ and $\boldsymbol{W}^{\boldsymbol{D}^{\boldsymbol{l}}} \in \mathbb{R}^{(h\times d)\times t}$ are the respective hypernetwork parameters. We additionally define the hypernetwork $h^l_{LN}(.)$ that computes the layer normalization parameters:

$$(\boldsymbol{\gamma}^{\boldsymbol{l}}_{\boldsymbol{\tau}},\boldsymbol{\beta}^{\boldsymbol{l}}_{\boldsymbol{\tau}}) := h^l_{LN}(\boldsymbol{I}_{\boldsymbol{\tau}}) = \left(\boldsymbol{W}^{\gamma^l},\boldsymbol{W}^{\beta^l}\right)\boldsymbol{I}_{\boldsymbol{\tau}}, \tag{4.7}$$

where $\boldsymbol{W}^{\boldsymbol{\gamma}^{\boldsymbol{l}}} \in \mathbb{R}^{h\times t}$ and $\boldsymbol{W}^{\boldsymbol{\beta}^{\boldsymbol{l}}} \in \mathbb{R}^{h\times t}$.

4.2.4 HYPERFORMER++

A downside of introducing a separate hypernetwork in each layer of the Transformer is that it increases the overall number of parameters. We, therefore, propose to share hypernetworks across transformer layers. By having a shared hypernetwork that is reusable, this strategy results in a substantial reduction in the number of parameters. However, reapplying the same hypernetwork across all the layers introduces weight sharing across target parameters, which may not be desirable. To allow for a flexible parameterization of task conditional adapters/layer normalization, for a transformer of L layers, we introduce a set of *layer id* embeddings $\mathcal{I} = \{\boldsymbol{l_i}\}^L_{i=1}$, and *adapter position* embeddings $\mathcal{P} = \{\boldsymbol{p_j}\}^2_{j=1}$, which specify the position of adapter layers in each transformer block (after the attention layer or feed-forward layer), which are used as additional inputs to the hypernetworks. For simplicity, we consider $\boldsymbol{l_i} \in \mathbb{R}^t$, $\boldsymbol{p_j} \in \mathbb{R}^t$, and $\boldsymbol{z_\tau} \in \mathbb{R}^t$. We feed a concatenation of $(\boldsymbol{z_\tau},\boldsymbol{l_i},\boldsymbol{p_j})$ to a similar task projector network h'_I as in Eq. (4.5):

$$\boldsymbol{I_\tau} = h'_I(\boldsymbol{z_\tau},\boldsymbol{l_i},\boldsymbol{p_j}), \tag{4.8}$$

which is then followed by a shared layer normalization to compute final task embeddings $\boldsymbol{I_\tau} \in \mathbb{R}^t$ to the hypernetwork. This way, the hypernetwork is able to produce distinct weights for each task, adapter position, and layer of a transformer. Furthermore, layer id and adapter position embeddings are parameters that are learned via back-propagation, allowing us to train the whole model end-to-end conveniently.

4.3 Experiments

Datasets Following Raffel et al. (2020), we evaluate the performance of the models on the GLUE benchmark (Wang et al., 2019c). This benchmark covers multiple tasks of paraphrase detection (MRPC, QQP), sentiment classification (SST-2), natural language inference (MNLI, RTE, QNLI), and linguistic acceptability (CoLA).[3] The original test sets are not publicly available, and following Zhang et al. (2021b), for datasets fewer than 10K samples (RTE, MRPC, STS-B, CoLA), we divide the original validation set in half, using one half for validation and the other for the test. For the other larger datasets, we split 1k samples from the training set as our validation data and test on the original validation set.

Experimental details We use the HuggingFace implementation (Wolf et al., 2020) of the T5 model (Raffel et al., 2020). We fine-tune all models with a constant learning rate of 0.0003 and

[3]Following Raffel et al. (2020); Devlin et al. (2019), as a common practice, due to the adversarial nature of WNLI with respect to the training set, we do not experiment with WNLI.

Model	#Total params	#Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
					Single-Task Training						
$T5_{SMALL}$	8.0×	100%	**46.81**	**90.47**	**86.21/90.67**	**91.02/87.96**	**89.11/88.70**	**82.09**	**90.21**	**59.42**	**82.06**
$Adapters_{SMALL}$ ❣	1+8×0.01	**0.74%**	40.12	89.44	85.22/89.29	90.04/86.68	83.93/83.62	81.58	89.11	55.80	79.53
$T5_{BASE}$	8.0×	100%	54.85	92.19	**88.18/91.61**	**91.46/88.61**	**89.55/89.41**	**86.49**	91.60	67.39	84.67
$Adapters_{BASE}$ ❣	1+8×0.01	**0.87%**	**59.49**	**93.46**	88.18/91.55	90.94/88.01	87.44/87.18	86.38	**92.26**	**68.84**	**84.88**
					Multi-Task Training						
$T5_{SMALL}$ ♠	1.0×	12.5%	50.67	**91.39**	84.73/88.89	**89.53/86.31**	**88.70/88.27**	81.04	89.67	59.42	81.69
$Adapters\dagger_{SMALL}$	1.05×	0.68%	39.87	90.01	88.67/91.81	88.51/84.77	88.15/87.89	79.95	89.60	60.14	80.85
$HYPERFORMER_{SMALL}$	1.45×	5.80%	47.64	**91.39**	**90.15/92.96**	88.68/85.08	87.49/86.96	**81.24**	**90.39**	65.22	82.47
$HYPERFORMER{+}{+}_{SMALL}$	1.04×	**0.50%**	**53.96**	90.59	84.24/88.81	88.44/84.46	87.73/87.26	80.69	**90.39**	**71.01**	**82.51**
$T5_{BASE}$ ♠	1.0×	12.5%	54.88	92.54	90.15/93.01	**91.13/88.07**	88.84/88.53	85.66	92.04	75.36	85.47
$Adapters\dagger_{BASE}$	1.07×	0.82%	61.53	93.00	90.15/92.91	90.47/87.26	89.86/89.44	86.09	**93.17**	70.29	85.83
$HYPERFORMER_{BASE}$	1.54×	6.86%	61.32	93.80	**90.64/93.33**	90.13/87.18	89.55/89.03	**86.33**	92.79	**78.26**	**86.58**
$HYPERFORMER{+}{+}_{BASE}$	1.02×	**0.29%**	**63.73**	**94.03**	89.66/92.63	90.28/87.20	**90.00/89.66**	85.74	93.02	75.36	86.48

Table 4.1: Performance of all models on the GLUE tasks. For each method, we report the total number of parameters across all tasks and the number of parameters that are trained for each task as a multiple and proportion respectively of the corresponding single-task T5 model. For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy. Adapters† refers to our proposed variant of adapters with shared layer normalizations. Our HYPERFORMER++ obtains a better score on average compared to full fine-tuning and Adapters†, while being more parameter-efficient. ♠: Our re-implementation of Raffel et al. (2020), ❣: Applying method of Houlsby et al. (2019) on T5. Bold fonts indicate the best results in each block.

following Raffel et al. (2020), we use $2^{18} = 262144$ steps in all experiments. We save a checkpoint every 1000 steps for all models (see also §4.7). Raffel et al. (2020) report the results based on the best checkpoint for each task independently. In contrast, we focus on the more realistic setting where we report the results on a single checkpoint with the highest average validation performance across all tasks. The hyperparameters are selected in the same manner. In contrast to prior work (Houlsby et al., 2019), we do not learn a separate output layer for each task but instead share a frozen output layer for all the tasks, which makes our setting more parameter-efficient than prior work and is an advantage of multi-task learning with encoder-decoder models.[4]

Baselines We compare to the strong adapter baseline (Houlsby et al., 2019). Following Houlsby et al. (2019), we add adapters modules for each task after the two feed-forward modules in each transformer block of the T5 model. As suggested in Houlsby et al. (2019), we train the layer normalization parameters inside the T5 model, per task. We refer to this method as *Adapters*. We additionally propose a variant of this model, in which we share all layer normalization parameters (T5 and adapters) across all tasks. We refer to this model as *Adapters*†. We compare our models to the state-of-the-art T5 model, in which we fine-tune all parameters of the model on all tasks. We refer to this method as $T5_{SMALL}$/$T5_{BASE}$ in experiments.

[4]According to our initial experiments, fine-tuning the final output layer did not improve performance for adapter-based methods.

Sampling tasks During training, we sample tasks with conventional temperature-based sampling with temperature $T = 10$ for all methods. We sample different tasks proportional to $p_\tau^{1/T}$ where $p_\tau = \frac{N_\tau}{\sum_{i=1}^{T} N\tau}$ and N_τ is the number of training samples for the τ-th task. We did not experiment with more complex sampling strategies (Raffel et al., 2020) or tuning of T.

4.3.1 Results on the GLUE Benchmark

Table 4.1 shows the results on GLUE for single-task and multi-task training. We experiment with reduction factors of $r = \{8,16,32\}$ for all adapter-based methods, where $r = \frac{h}{d}$. We report the results both with T5_{SMALL} (6 layers and 60M parameters) and T5_{BASE} models (12 layers and 222M parameters).

Overall, our proposed HYPERFORMER++ obtains strong gains over Adapters (82.51 versus 79.53 for T5_{SMALL} and 86.48 versus 84.88 for T5_{BASE}) while being more parameter-efficient.

Our variant of Adapters†, which shares layer norms across tasks, outperforms prior work (Houlsby et al., 2019), which does not share such information (80.85 versus 79.53 for T5_{SMALL} and 85.83 versus 84.88 for T5_{BASE}). This demonstrates that in encoder-decoder models such as T5 more sharing of information across tasks is beneficial.

Our proposed HYPERFORMER obtains consistent improvement over our proposed Adapters† method. We attribute this improvement to the ability to learn the shared information across tasks through our hypernetworks. Interestingly, HYPERFORMER++ obtains similar performance as HYPERFORMER while being more than an order of magnitude more parameter-efficient. Adapter modules thus seem to be similar enough so that much of their information can be modeled by a single, appropriately conditioned network.

Compared to single-task fine-tuning of all parameters, our methods on average improve the results by 0.45 for T5_{SMALL} and 1.81 for T5_{BASE} with substantial improvement on low-resource datasets like CoLA (63.73 versus 54.85) and RTE (75.36 versus 67.39) due to shared hypernetworks that capture the shared information and enable positive transfer effects.

We also report the total number of parameters and trainable parameters for all methods in Table 4.1. For adapter-based methods, the number of parameters varies based on the adapter size (we report all numbers with $r = 32$). The multiple in terms of the number of parameters of $\text{HYPERFORMER++}_{\text{BASE}}$ with regard to T5_{BASE} is $1.02\times$ with only 0.29% trainable parameters per task. Note that by keeping the output layer frozen for $\text{Adapters}_{\text{SMALL}}$ and $\text{Adapters}_{\text{BASE}}$, they require $5.51\times$ and $2.53\times$ fewer parameters respectively compared to a direct application of prior work (Houlsby et al., 2019). Despite using more efficient baselines, compared to $\text{Adapters}_{\text{BASE}}$, $\text{HYPERFORMER++}_{\text{BASE}}$ requires $3\times$ fewer trainable parameters.

4.3.2 Few-shot Domain Transfer

Finally, we assess how well a trained HYPERFORMER can generalize to new tasks. We evaluate performance on 5 tasks and 7 datasets. In particular, we consider 1) the natural language inference (NLI) datasets SciTail (Khot et al., 2018), and CB (De Marneffe et al., 2019) from SuperGLUE (Wang et al., 2019b) 2) the question answering (QA) dataset BoolQ (Clark et al., 2019a); 3) the sentiment analysis datasets IMDB (Maas et al., 2011) and Yelp Polarity (Zhang et al., 2015); and 4) the paraphrase detection dataset PAWS (Baldridge et al., 2019); 5) the question classification dataset TREC (Li and Roth, 2002).

For CB and BoolQ, since test sets are not available, we divide the validation sets in half, using one half for validation and the other for testing. For Yelp polarity, TREC, and IMDB, since validation sets are not available, we similarly divide the test sets to form validation sets. For the rest, we report on the original test sets.

We consider the models trained on GLUE reported in Table 4.1 and evaluate them on the test set after the few-shot fine-tuning on each target training data. For Adapters† and our method, we use the adapter and the task embedding respectively trained on the most similar GLUE task for initialization, i.e. MNLI for NLI, QNLI for QA, SST-2 for sentiment analysis, and QQP for paraphrase detection. Following prior evidence of positive transfer from NLI to other tasks (Conneau and Kiela, 2018; Yin et al., 2020; Phang et al., 2018), we initialize the out-of-domain TREC from MNLI. We show the results of full fine-tuning of all model's parameters, Adapters†, and HYPERFORMER++[5] in Table 4.2. Our method significantly surpasses the baselines on the majority of settings.

4.3.3 Low-resource Fine-tuning

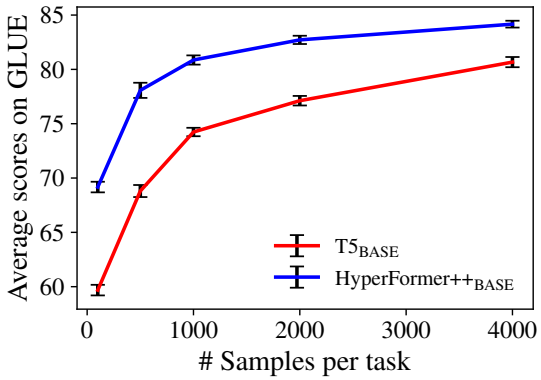

Figure 4.2: Results on GLUE for the various number of training samples per task (100,500,1000,2000,4000). We show mean and standard deviation across 5 seeds.

Given that our model HYPERFORMER++$_{\text{BASE}}$ has substantially fewer trainable parameters than T5$_{\text{BASE}}$,

[5]We finetune hypernetworks and task embeddings parameters. We also tried only fine-tuning the task embedding but found that this achieves lower performance in the few-shot setting and comparable performance with more samples.

Dataset	# Samples	$T5_{BASE}$	$Adapters^{\dagger}_{BASE}$	$HYPERFORMER{+}{+}_{BASE}$
		Natural Language Inference		
SciTail	4	79.60±3.3	79.54±2.8	**82.00**±4.9
	16	80.03±2.3	83.25±1.7	**86.55**±1.4
	32	81.97±1.3	85.06±1.1	**85.85**±1.4
	100	84.04±0.7	88.22±1.3	**88.52**±0.7
	500	88.07±0.7	91.27±0.8	**91.44**±0.6
	1000	88.77±1.0	91.75±0.8	**92.34**±0.5
	2000	91.01±1.0	92.72±0.5	**93.40**±0.2
CB	4	57.78±10.9	51.11±9.2	**60.74**±16.66
	16	**77.04**±7.2	74.81±5.4	76.29±4.45
	32	80.0±7.6	74.81±5.9	**81.48**±6.2
	100	85.93±5.4	80.74±7.6	**87.41**±2.96
	250	85.19±4.7	86.67±5.0	**89.63**±4.32
		Question Classification		
TREC	4	28.11±5.9	23.61±7.7	**28.85**±6.9
	16	40.08±12.6	43.45±14.0	**49.40**±9.5
	32	62.49±6.2	59.6±7.0	**68.94**±7.5
	100	87.79±0.7	78.07±3.8	**88.42**±1.7
	500	93.57±1.3	93.65±1.7	**94.78**±1.4
	1000	95.5±0.9	96.06±0.4	**96.72**±1.3
	2000	96.87±1.3	**97.03**±0.7	96.92±0.9
		Question Answering		
BoolQ	4	50.49±11.1	**53.48**±2.8	48.03±4.8
	16	**56.50**±7.1	51.37±6.5	50.21±7.9
	32	**58.43**±4.9	54.52±5.1	58.37±3.7
	100	60.10±2.4	58.60±1.6	**62.03**±2.0
	500	66.49±1.2	66.72±0.7	**70.04**±1.4
	1000	69.01±1.1	70.21±1.3	**72.35**±1.7
	2000	71.58±0.8	73.60±0.8	**74.94**±0.6
		Sentiment Analysis		
IMDB	4	77.23±3.0	81.55±1.9	**81.77**±1.8
	16	82.74±1.7	82.54 ±1.0	**84.06**±0.7
	32	83.42±1.0	83.39 ±0.8	**84.64**±0.4
	100	84.58±0.6	83.35 ±0.8	**84.74**±0.4
	500	84.99±0.3	85.37±0.5	**86.00**±0.2
	1000	85.50±0.1	86.27±0.4	**86.37** ±0.4
	2000	86.01±0.2	86.57±0.2	**86.60**±0.1
Yelp polarity	4	76.85±14.3	81.37±13.1	**90.25**±1.0
	16	87.84±1.5	**91.08**±0.2	90.36±1.2
	32	89.22±0.7	91.09±0.5	**91.15**±0.5
	100	90.19±0.7	90.15±0.7	**91.06**±0.6
	500	90.92±0.2	91.52±0.2	**92.09**±0.4
	1000	91.32±0.2	92.26±0.6	**92.50**±0.2
	2000	91.68±0.1	92.36±0.4	**92.70**±0.1
		Paraphrase Detection		
PAWS	4	53.89±3.6	**55.69**±9.0	55.58±7.5
	16	54.18±1.0	63.38±5.3	**72.71**±1.1
	32	55.23±3.2	68.78±1.5	**73.39**±2.1
	100	71.51±2.4	73.82±1.6	**78.24**±2.1
	500	82.81±1.0	85.36±0.6	**86.3**±1.1
	1000	85.67±0.7	87.89±0.6	**89.12**±0.5
	2000	88.33±0.6	90.41±0.6	**90.87**±0.3

Table 4.2: Few-shot domain transfer results of the models trained on GLUE averaged across 5 seeds. We compute accuracy for all datasets.

we investigate whether it generalizes better in a low-resource setting. We subsample each individual task in GLUE for varying training sizes. We train the models for 15,000 steps, which we found to be sufficient to allow them to converge. Figure 4.2 shows the results. HYPERFORMER++$_{\text{BASE}}$ substantially improves results with limited training data, indicating more effective fine-tuning in this regime.

4.4 Analysis

4.4.1 Parameter Efficiency

In this section, we compare the number of parameters of HYPERFORMER++ with Adapters.

Adapters parameters The standard setting (Houlsby et al., 2019) employs two adapters per layer for each task. Each adapter layer has $2hd$ parameters for projection matrices ($\boldsymbol{U}_{\tau}^{l}$ and $\boldsymbol{D}_{\tau}^{l}$) and $2h$ parameters for the layer normalization. The total number of parameters for Adapters for L Transformer layers in both an encoder and a decoder across T tasks is, therefore, $4TL(2hd+2h)$, which scales linearly with the number of tasks times the number of layers.

HYPERFORMER++ parameters: Our approach learns a task feature embedding per task, consisting of Tt parameters. We additionally employ layer id and adapter position embeddings in the encoder and decoder, which require $2(2+L)t$ parameters, with a fixed embedding size of t for all these feature embeddings. We consider a separate task projector networks h_I' for encoder and decoder, which is in both cases a two-layer MLP, consisting of a total of $2(3te+et)$ parameters, where $e=128$ is the hidden dimension for the task-projector network. Our hypernetwork for adapters in encoder/decoder consists of $2(2thd)$ parameters and our layer normalization hypernetwork consists of $2(2th)$ parameters. In total, this results in $\underbrace{t(T+4+2L)}_{\text{Task features}}+\underbrace{8te+2t(2hd+2h)}_{\text{Hypernetworks}}$ parameters. The total number of parameters for hypernetworks remains constant, while the task feature parameters scale with the number of tasks or layers times t, where $t=64$ in our experiments.

In settings with a large number of layers and a large number of tasks, since $t \ll 2hd+2h$ and $T+L \ll TL$, our method is much more parameter-efficient compared to Adapters. In the current setting, the term hd is the largest term, and the factor $2TL$ for Adapters is larger than the factor t for HYPERFORMER++.

4.4.2 Do Extra Parameters Make a Difference?

While our HYPERFORMER++ is more parameter-efficient than the baselines, the number of parameters of HYPERFORMER per task is higher compared to Adapters†. To confirm that the improvements of HYPERFORMER are due to its capability of sharing information across tasks and not the number of parameters, as an ablation, we run the Adapters† with $r=\{2,4\}$ and choose the model performing the best on the validation set. This allows Adapters† to have a higher number of parameters compared to

Model	GLUE	#Total params	#Trained params/task
Adapters† $_{\text{SMALL}}$	80.97	1.83x	10.44%
HYPERFORMER $_{\text{SMALL}}$	82.47	1.45x	5.80 %
Adapters† $_{\text{BASE}}$	85.84	2.02x	12.73%
HYPERFORMER $_{\text{BASE}}$	86.58	1.54x	6.86%

Table 4.3: Averaged test results on GLUE for HYPERFORMER and Adapters†, where Adapters† has a higher number of parameters compared to HYPERFORMER.

Model variant	GLUE
HYPERFORMER$_{\text{SMALL}}$	82.47
– Adapter blocks	68.37
– Conditional layer norm	79.83
– Task projector	81.56
– T5 Layer norm	81.29
– Conditional layer norm, T5 Layer norm	78.92

Table 4.4: Impact when removing different components of our framework. We report the average results on GLUE.

HYPERFORMER. We report the results in Table 4.3 and compare them with results of HYPERFORMER in Table 4.1. The results demonstrate that even with an increased number of parameters, Adapters† is not able to reach the performance of HYPERFORMER, and HYPERFORMER performs substantially better.

4.4.3 Impact of the Framework Components

We investigate the impact of the components of our framework including: (1) task conditional adapter blocks; (2) task conditional layer normalization; (3) task projection network; (4) fine-tuning of layer normalizations in the T5 model; (5) task conditional layer normalization in adapter modules and fine-tuning of layer normalizations inside the T5 model. We consider our small model of Table 4.1 and train different variants of it. Table 4.4 shows the results on GLUE, demonstrating that each component of the model contributes positively to its final performance.

4.4.4 Visualization of Task Embeddings

To analyze what HYPERFORMER++$_{\text{BASE}}$ has learned about the relations between different tasks, we visualize the learned task embeddings for the models trained with the largest number of samples in Table 4.1 and 4.2. Figure 4.3 illustrates the 2D vector projections of task embeddings using PCA (Wold et al., 1987). Interestingly, the observed groupings correspond to similar tasks. This shows that learned task embeddings by HYPERFORMER++$_{\text{BASE}}$ are meaningful. For CB, an NLI dataset despite being initialized from MNLI, after few-shot training the task embedding is closest to RTE, another NLI

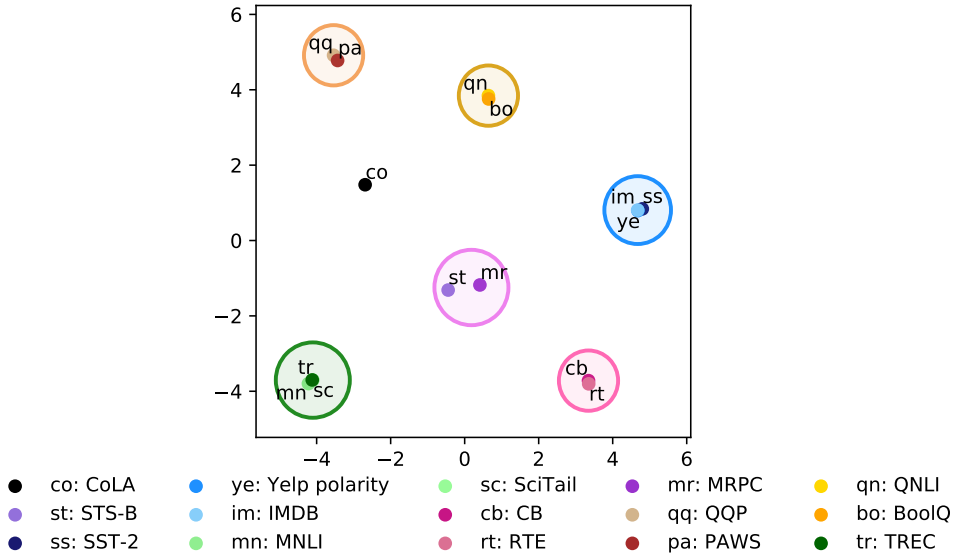

Figure 4.3: Visualization of learned task embeddings by HYPERFORMER++$_{\text{BASE}}$.

dataset. This is plausible as premises and hypotheses in both the discourse-based CB and the news and Wikipedia-based RTE are more complex compared to MNLI. The sentence similarity dataset STS-B is grouped close to the MRPC paraphrase dataset. CoLA, which focuses on linguistic acceptability is very different from other tasks and is not grouped with any of the observed task embeddings. In addition, the task embeddings for 1) all the sentiment analysis datasets namely SST-2, Yelp polarity, and IMDB; 2) the two large-scale NLI datasets namely MNLI and SciTail; 3) question answering datasets, i.e. BoolQ and QNLI; and 4) paraphrase datasets namely QQP and PAWS are each grouped together.

4.5 Related Work

Multi-task learning Multi-task learning, i.e., learning a unified model to perform well on multiple different tasks, is a challenging problem in NLP. It requires addressing multiple challenges such as catastrophic forgetting, and handling disproportionate task sizes resulting in a model overfitting in low-resource tasks while underfitting in high-resource ones (Arivazhagan et al., 2019). Liu et al. (2019a) proposed Multi-Task Deep Neural Network (MTDNN) for learning from multiple NLU tasks. Although MTDNN obtains impressive results on GLUE, it applies multi-task learning as a form of pretraining followed by task-specific fine-tuning. Concurrently with us, Tay et al. (2021) propose a multi-task learning method by training task-conditioned hyper networks; however, their method is 43x less parameter efficient compared to ours. In another line of research, Clark et al. (2019c) proposed to learn multi-task models with knowledge distillation. Houlsby et al. (2019) trained adapters for each task separately, keeping the model fixed. Stickland and Murray (2019) share the model parameters across tasks and introduce task-specific adapter parameters, which is more parameter-inefficient than our method.

Hypernetworks and contextual parameter generation Our work is closely related to hypernetworks (Ha et al., 2017). In a continual learning setup, where tasks are learned sequentially, Oswald et al. (2020) proposed a task-conditioned hypernetwork to generate all the weights of the target model. Our method is substantially more efficient as we do not generate all the weights of the target model but a very small number of parameters for adapter modules to allow the model to adapt to each individual task efficiently. Similarly, Jin et al. (2020) generate the full model from task-specific descriptions in different domains whereas we efficiently generate only small adapter modules for each task.

Prior work also proposed meta-learning or Bayesian approaches to generate softmax layer parameters for new settings (Bansal et al., 2020; Ponti et al., 2020). Meta-learning approaches are notoriously slow to train. In addition, generating softmax parameters requires a substantially higher number of parameters, leaves the method unable to adapt the lower layers of the model, and restricts their application to classification tasks.

In contemporaneous work, Üstün et al. (2020) proposed a multilingual dependency parsing method based on adapters and contextual parameter generator networks (Platanios et al., 2018) where they generate adapter parameters conditioned on trained input language embeddings. Their study is limited to multilingual dependency parsing, while our work studies multi-task learning and applies to several tasks thanks to the general sequence-to-sequence nature of our model. Moreover, their number of trainable parameters is $2.88\times$ larger than their base model since they employ a contextual parameter generator in each layer. In contrast, we use a single compact hypernetwork allowing us to efficiently condition on multiple tasks and layers of a transformer model.

4.6 Conclusion

In this chapter, we propose a parameter-efficient method for multi-task fine-tuning. Our approach is to train shared hypernetworks to generate task-specific adapters conditioned on the task, layer id, and adapter position embeddings. The shared hypernetworks capture the knowledge across tasks and enable positive transfer to low-resource and related tasks, while task-specific layers allow the model to adapt to each individual task. Extensive experiments show that our method obtains strong improvement over multi-task learning on the GLUE benchmark, and substantially improves the in-domain task generalization.

Appendix

4.7 Experimental Details

Computing infrastructure We run the experiments in Table 4.1 on 4 GPUs, and the rest of the experiments on 1 GPU on a heterogeneous cluster with Tesla V100, Tesla A100, Tesla P4, and GTX1080ti GPUs.

Hyperparameters We use a batch size of 64 for T5_{SMALL} and 32 for T5_{BASE} to fit the GPU memory. We set the dimension of the task feature embedding ($\boldsymbol{z_\tau}$) to $t'=512$, and the dimension of the task embedding ($\boldsymbol{I_\tau}$) to $t=64$. For low-resource fine-tuning in §4.3.3, we use reduction factors of $\{16,32,64\}$.

Data pre-processing We download all datasets from the HuggingFace Datasets library Lhoest et al. (2021). Following Raffel et al. (2020), we cast all datasets into a sequence-to-sequence format, and re-cast STS-B as a 21-class classification task by rounding its target scores to their nearest increment of 0.2.

Performance evaluation Table 4.5 and 4.6 present the efficiency evaluation in terms of memory, and time for all the methods measured on the GLUE benchmark. We report the time for 1000 training steps.

Our approach has several attractive properties. Our $\text{HYPERFORMER++}_{\text{BASE}}$ approach offers a much better memory usage with low-overhead, while $\text{HYPERFORMER}_{\text{BASE}}$ and T5_{BASE} cause substantial memory overhead. In dealing with large-scale transformer models like T5, efficient memory usage is of paramount importance. Second, in terms of training time, our method is much faster than $\text{Adapters}\dagger_{\text{BASE}}$. Relative to T5_{BASE}, $\text{HYPERFORMER++}_{\text{BASE}}$ increases the training time by 30.49%, while $\text{Adapters}\dagger_{\text{BASE}}$ causes the substantial training time overhead of 84.93%.

Model	Memory	Δ%
$T5_{BASE}$	7.76 (GB)	-
$Adapters^{\dagger}_{BASE}$	5.95 (GB)	-23.32%
$HYPERFORMER_{BASE}$	7.60 (GB)	-2.06%
$HYPERFORMER++_{BASE}$	5.81 (GB)	-25.13

Table 4.5: The required memory for all methods. Δ% is the relative difference with respect to $T5_{BASE}$.

Model	Time	Δ%
$T5_{BASE}$	5.51 (min)	-
$Adapters^{\dagger}_{BASE}$	10.19 (min)	84.93%
$HYPERFORMER_{BASE}$	7.92 (min)	43.74%
$HYPERFORMER++_{BASE}$	7.19 (min)	30.49%

Table 4.6: Training time for all methods. Δ% is the relative difference with respect to $T5_{BASE}$.

Impact of adapter's bottleneck size on the performance Similar to (Houlsby et al., 2019), adapter's reduction factor needs to be set per dataset. Table 4.7 shows the validation performance of HYPERFORMER++ on the GLUE tasks for different adapters' reduction factors. While the pattern may not be always consistent, generally, smaller datasets seem to benefit more from smaller bottleneck size, i.e., less parameters for adapters, while the opposite is the case for larger datasets, which require more modeling capacity.

Model	r	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
$HYPERFORMER++_{SMALL}$	8	42.13	98.60	82.76/87.72	90.69/87.55	84.92/84.18	82.3	95.40	78.83	83.19
$HYPERFORMER++_{SMALL}$	16	42.60	97.8	84.73/89.12	88.99/85.33	85.69/85.12	81.96	93.69	75.91	82.81
$HYPERFORMER++_{SMALL}$	32	49.90	96.00	83.74/88.50	89.29/85.79	85.99/85.41	81.28	91.79	72.99	82.79
$HYPERFORMER++_{BASE}$	8	54.86	97.30	88.18/91.55	94.59/92.91	89.77/89.69	85.89	96.10	84.67	87.77
$HYPERFORMER++_{BASE}$	16	53.83	98.00	88.18/91.61	94.89/93.33	90.12/89.65	85.94	96.50	83.94	87.82
$HYPERFORMER++_{BASE}$	32	55.58	97.20	89.66/92.42	93.19/91.08	88.96/88.57	85.82	94.19	81.75	87.13

Table 4.7: Validation performance of HYPERFORMER++ on the GLUE tasks for different reduction factors $r = \{8,16,32\}$. For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy.

5 COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers

Adapting large-scale pretrained language models to downstream tasks via fine-tuning is the standard method for achieving state-of-the-art performance on NLP benchmarks. However, fine-tuning all weights of models with millions or billions of parameters is sample-inefficient, unstable in low-resource settings, and wasteful as it requires storing a separate copy of the model for each task. Recent work has developed *parameter-efficient* fine-tuning methods, but these approaches either still require a relatively large number of parameters or underperform standard fine-tuning. In this chapter, we propose COMPACTER, a method for fine-tuning large-scale language models with a better trade-off between task performance and the number of trainable parameters than prior work. COMPACTER accomplishes this by building on top of ideas from adapters, low-rank optimization, and parameterized hypercomplex multiplication layers.

Specifically, COMPACTER inserts task-specific weight matrices into a pretrained model's weights, which are computed efficiently as a sum of Kronecker products between shared "slow" weights and "fast" rank-one matrices defined per COMPACTER layer. By only training 0.047% of a pretrained model's parameters, COMPACTER performs on par with standard fine-tuning on GLUE and outperforms standard fine-tuning on SuperGLUE and low-resource settings.[1]

5.1 Introduction

> With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.
>
> John von Neumann

State-of-the-art pretrained language models (PLMs) in natural language processing (NLP) have used heavily over-parameterized representations consisting of hundreds of millions or billions of parameters to achieve success on a wide range of NLP benchmarks (Devlin et al., 2019; Raffel et al., 2020; Liu et al., 2019b). These models are generally

[1] Our code is publicly available at `https://github.com/rabeehk/compacter`.

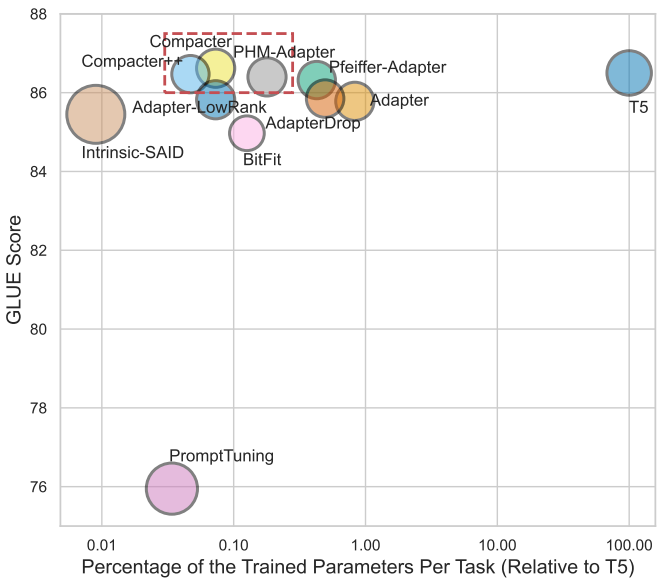

Figure 5.1: The average score on GLUE (y axis), percentage of trainable parameters per task (x axis, in log scale), and memory footprint (size of the circles) of different methods.

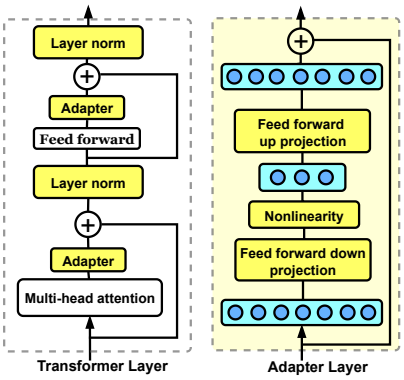

Figure 5.2: Left: Adapter integration in a pretrained transformer model. Right: Adapter architecture. Following Houlsby et al. (2019), we include adapters after the attention and feed-forward modules. During training, we only update layer normalizations and adapters (shown in yellow), while the pretrained model is fixed.

applied to downstream tasks via fine-tuning (Howard and Ruder, 2018), which requires updating *all* parameters and storing one copy of the fine-tuned model per task. This causes substantial storage and deployment costs and hinders the applicability of large-scale PLMs to real-world applications. Additionally, fine-tuning of over-parameterized models on low-resource datasets has been shown to be subject to instabilities and may lead to poor performance (Peters et al., 2019; Dodge et al., 2020).

Inspired by John von Neumann's quotation, we ask, given that we have already learned general-purpose language representations via a PLM (i.e. we have fit our elephant), how many more parameters do we need to reach state-of-the-art performance on standard NLP tasks. Specifically, we aim to develop practical, memory-efficient methods that train a minimum set of parameters while achieving performance on par or better than full fine-tuning for state-of-the-art NLP models.

Recent literature has introduced *parameter-efficient* fine-tuning methods. These approaches generally keep the pretrained model's parameters fixed and introduce a set of trainable parameters per task, trading off the number of trainable parameters with task performance. At one end of the spectrum, *prompts*, i.e. natural language descriptions of a task, together with demonstrations have been used to achieve reasonable performance *without any* parameter updates on some benchmarks (Brown et al., 2020) but their performance generally lags behind fine-tuned models. They also require huge models to work well but choosing good prompts becomes harder with larger model sizes (Perez et al., 2021). *Soft prompt* methods treat prompts as trainable continuous parameters, which are prepended to the inputs at the input layer or intermediate layers (Li and Liang, 2021; Hambardzumyan et al., 2021; Lester et al., 2021). Such methods, however, often require large models to achieve good performance

and are very sensitive to initialization and unstable during training.

The theoretically motivated *low-rank* methods train a small number of parameters that lie in a low-dimensional subspace using random projections (Li et al., 2018; Aghajanyan et al., 2021). However, storing the random projection matrices causes substantial memory overhead and leads to slow training times. At the other end of the spectrum, *adapter* methods (Rebuffi et al., 2018; Houlsby et al., 2019) that insert trainable transformations at different layers of the pretrained model require more parameters than the aforementioned approaches but are more memory-efficient and obtain performance comparable to full fine-tuning (Houlsby et al., 2019; Lin et al., 2020).

In this work, we propose COMPACTER, a method for fine-tuning large-scale language models with an excellent trade-off between the number of trainable parameters, task performance, and memory footprint, compared to existing methods (see Figure 5.1). COMPACTER builds on ideas from adapters (Houlsby et al., 2019), low-rank methods (Li et al., 2018), as well as recent hypercomplex multiplication layers (Zhang et al., 2021a). Similar to adapters, COMPACTER inserts task-specific weight matrices into a pretrained model's weights. Each COMPACTER weight matrix is computed as the sum of Kronecker products between shared "slow" weights and "fast" rank-one matrices defined per COMPACTER layer (see Figure 5.3). As a result, COMPACTER achieves a parameter complexity of $\mathcal{O}(k+d)$ compared to $\mathcal{O}(kd)$ for regular adapters, where the adapters are of size $k \times d$. In practice, COMPACTER trains 0.047% of a PLM's parameters. On the standard GLUE (Wang et al., 2019c) and SuperGLUE (Wang et al., 2019b) benchmarks, COMPACTER outperforms other parameter-efficient fine-tuning methods and obtains performance on par or better than full fine-tuning. On low-resource settings, COMPACTER outperforms standard fine-tuning.

In summary, in this chapter, we make the following contributions: **1)** We propose COMPACTER (**Compact** Adapt**er**) layers, a parameter-efficient method to adapt large-scale language models. **2)** We show that COMPACTER obtains strong empirical performance on GLUE and SuperGLUE. **3)** We demonstrate that COMPACTER outperforms fine-tuning in low-resource settings. **4)** We provide a parameter complexity analysis of COMPACTER, showing that it requires dramatically fewer parameters than adapters and fine-tuning. **5)** We provide a systematic evaluation of recent parameter-efficient fine-tuning methods in terms of training time and memory consumption.

5.2 Background

We start by introducing the required background on the Kronecker product and adapter layers (Houlsby et al., 2019; Rebuffi et al., 2018).

5.2.1 Kronecker Product

The Kronecker product between matrix $\boldsymbol{A} \in \mathbb{R}^{m \times f}$ and $\boldsymbol{B} \in \mathbb{R}^{p \times q}$, denoted by $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{mp \times fq}$, is mathematically defined as:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1f}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mf}\boldsymbol{B} \end{pmatrix}, \tag{5.1}$$

where a_{ij} shows the element in the i^{th} row and j^{th} column of $\boldsymbol{A}$.

5.2.2 Adapter Layers

Recent work has shown that fine-tuning *all* parameters of a language model can lead to a sub-optimal solution, particularly for low-resource datasets (Peters et al., 2019). As an alternative, Rebuffi et al. (2018) and Houlsby et al. (2019) propose to transfer a model to new tasks by inserting small task-specific modules called *adapter layers* within the layers of a pretrained model, as depicted in Figure 5.2. They then only train adapters and layer normalizations, while the remaining parameters of the pretrained model remain fixed. This approach allows pretrained language models to efficiently adapt to new tasks.

Each layer of a transformer model is composed of two primary modules: a) an attention block, and b) a feed-forward block. Both modules are followed by a skip connection. As shown in Figure 5.2, Houlsby et al. (2019) suggest to insert an adapter layer after each of these blocks before the skip connection.

Adapters are bottleneck architectures. By keeping the output dimension similar to their input, they cause no change to the structure or parameters of the original model. The adapter layer A^l for layer l consists of a down-projection, $\boldsymbol{D^l} \in \mathbb{R}^{k \times d}$, GeLU non-linearity (Hendrycks and Gimpel, 2016), and up-projection $\boldsymbol{U^l} \in \mathbb{R}^{d \times k}$, where k is the input dimension, and d is the bottleneck dimension for the adapter layer. Adapters are defined as:

$$A^l(\boldsymbol{x}) = \boldsymbol{U^l}(\text{GeLU}(\boldsymbol{D^l}(\boldsymbol{x}))) + \boldsymbol{x}, \tag{5.2}$$

where $\boldsymbol{x}$ is the input hidden state.

5.3 Method

In this section, we present COMPACTER, a compact and efficient way to adapt large-scale PLMs.

Problem formulation We consider the general problem of fine-tuning large-scale language models, where we are given the training data $\mathcal{D} = \{(\boldsymbol{x^i}, y^i)\}_{i=1}^{P}$ with P samples. We assume we are also given a large-scale pretrained language model $f_{\boldsymbol{\theta}}(.)$ parameterized by $\boldsymbol{\theta}$ that computes the output for input $\boldsymbol{x^i}$. Our goal is to fine-tune $f_{\boldsymbol{\theta}}(.)$ efficiently to enable the model to adapt to new tasks.

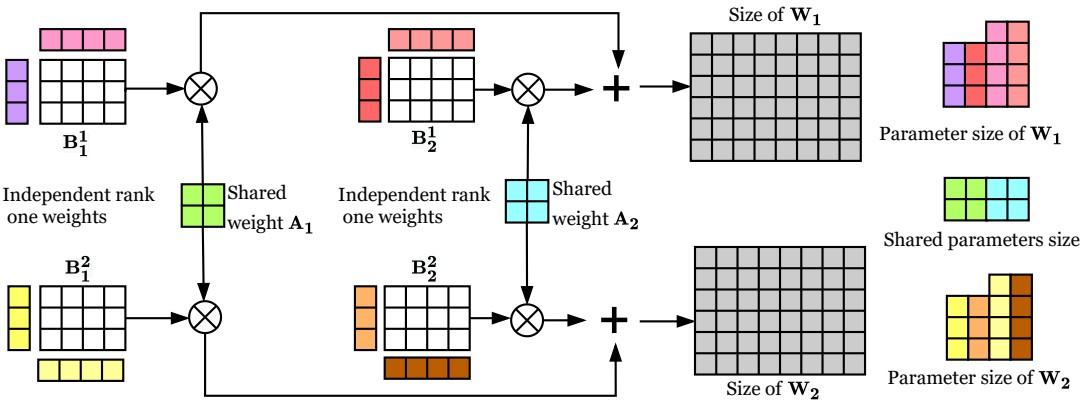

Figure 5.3: Illustration of generating weights of two different COMPACTER layers: $\boldsymbol{W_1} \in \mathbb{R}^{d\times k}$ (first row) and $\boldsymbol{W_2} \in \mathbb{R}^{d\times k}$ (second row). We generate $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ using $\boldsymbol{W_j} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i}^j = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes (\boldsymbol{s_i}^j \boldsymbol{t_i}^{j\top})$ (5.5), by computing the sum of Kronecker products of *shared* matrices $\boldsymbol{A_i}$ and *adapter-specific* matrices $\boldsymbol{B_i^j}$, with $i \in \{1,...,n\}$ and adapter index $j \in \{1,2\}$. We generate each $\boldsymbol{B_i^j}$ by multiplying independent rank one weights. In this example $n=2$, $d=6$, and $k=8$.

5.3.1 Compact and Efficient Adapter Layers

In this section, we introduce an efficient version of adapter layers, building on top of recent advances in *parameterized hypercomplex multiplication layers* (PHM) (Zhang et al., 2021a). To the best of our knowledge, we are the first to exploit PHM layers for efficient fine-tuning of large-scale transformer models. The PHM layer has a similar form as a fully-connected layer, which converts an input $\boldsymbol{x} \in \mathbb{R}^k$ to an output $\boldsymbol{y} \in \mathbb{R}^d$:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{5.3}$$

where $\boldsymbol{W} \in \mathbb{R}^{k\times d}$. The key difference is that in a PHM layer, $\boldsymbol{W}$ is learned as a sum of Kronecker products. Assume that k and d are both divisible by a user-defined hyperparameter $n \in \mathbb{Z}_{>0}$. Then, the matrix $\boldsymbol{W}$ in (5.3) is computed as the sum of n Kronecker products as follows:

$$\boldsymbol{W} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i}, \tag{5.4}$$

where $\boldsymbol{A_i} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n}\times\frac{d}{n}}$. The PHM layer has a parameter complexity of $\mathcal{O}(\frac{kd}{n})$, reducing parameters by at most $\frac{1}{n}$ (Zhang et al., 2021a) (see §5.4).

5.3.2 Beyond Hypercomplex Adapters

Prior work indicates that some of the information captured in pretrained models can be ignored for transfer (Zhang et al., 2021b; Chung et al., 2021). Similarly, redundancies have been observed in the information captured by adapters, with adapters in lower layers being less important (Houlsby et al., 2019). In addition, sharing adapters across layers leads to a comparatively small drop of performance

for some tasks (Rücklé et al., 2021). Motivated by these insights, we propose the following two extensions to make hypercomplex adapters more efficient.

Sharing information across adapters Sharing all adapter parameters across layers is overall too restrictive and is not able to perform on par with fine-tuning or using regular adapters (Rücklé et al., 2021); however, our decomposition of adapters into $\boldsymbol{A_i}$ and $\boldsymbol{B_i}$ matrices as in Eq. (5.4) allows us to be more flexible. Consequently, we divide our adaptation weights into *shared* parameters that capture general information useful for adapting to the target task and *adapter-specific* parameters that focus on capturing information relevant for adapting each individual layer. Specifically, we define $\boldsymbol{A_i}$ as shared parameters that are common across all adapter layers while $\boldsymbol{B_i}$ are adapter-specific parameters.

Low-rank parameterization Low-rank methods (Li et al., 2018; Aghajanyan et al., 2021) have demonstrated that strong performance can be achieved by optimizing a task in a low-rank subspace. Similarly, we hypothesize that a model can also be effectively adapted by learning transformations in a low-rank subspace. To this end, we propose to parameterize $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}$ as a low-rank matrix, which is the product of two low-rank weights $\boldsymbol{s_i} \in \mathbb{R}^{\frac{k}{n} \times r}$ and $\boldsymbol{t_i} \in \mathbb{R}^{r \times \frac{d}{n}}$, where r is the rank of the matrix.[2] Putting both extensions together, we propose the *low-rank* parameterized hypercomplex multiplication layer (LPHM):

$$\boldsymbol{W} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes \boldsymbol{B_i} = \sum_{i=1}^{n} \boldsymbol{A_i} \otimes (\boldsymbol{s_i} \boldsymbol{t_i}^{\top}). \tag{5.5}$$

In general, we set $r = 1$ so that $\boldsymbol{B_i}$ is a rank-one matrix. Depending on the complexity of the target task, r can be set to a higher value.[3] Figure 5.3 illustrates our method. Overall, the LPHM layer reduces complexity further to $\mathcal{O}(k + d)$ (see §5.4). The LPHM layer can also be seen as leveraging "slow" weights $\boldsymbol{A_i}$ that are shared across adapters and capture general information and "fast" weights $\boldsymbol{B_i}$ that learn adapter-specific information for adaptation of each individual layer (Wen et al., 2020).

COMPACTER Based on the above formulation, we introduce COMPACTER layers, which replace the down-projection and up-projection layers in adapters as follows:

$$A^l(\boldsymbol{x}) = \text{LPHM}^{U^l}(\text{GeLU}(\text{LPHM}^{D^l}(\boldsymbol{x}))) + \boldsymbol{x},$$

where the up-projection weights LPHM^{U^l} are computed as in (5.5), replacing the layer U^l in (5.2). Similarly, down-projection weights LPHM^{D^l} replace the layer D^l. While the two adapters in each layer of a transformer have their own $\boldsymbol{s_i}$ and $\boldsymbol{t_i}$ rank-one weights, we share the $\boldsymbol{A_i}$ across all layers and positions of the adapter layers.

5.4 Parameter Efficiency

In this section, we compare the number of parameters of COMPACTER with adapters.

[2] We do not factorize $\boldsymbol{A_i}$ as they are small, shared between all layers, and factorization hurts performance.

[3] If factors are over-parameterized, COMPACTER can be used for *overcomplete* knowledge distillation (Arora et al., 2018).

Adapters parameters In the standard setting, two adapters are added per layer of a transformer model (Houlsby et al., 2019). Each adapter layer consists of $2kd$ parameters for the down and up-projection matrices ($\boldsymbol{U^l}$, $\boldsymbol{D^l}$) respectively where k is the size of the input dimension and d is the adapter's bottleneck dimension. The total number of parameters for adapters for a transformer model with L layers of both an encoder and a decoder is, therefore, $2L(2kd)$, which scales linearly with all three variables.

PHM-ADAPTER parameters In the conventional PHM layer (Zhang et al., 2021a), as depicted in Eq. (5.4), parameters of $\boldsymbol{A_i} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{B_i} \in \mathbb{R}^{\frac{k}{n}\times\frac{d}{n}}$ define the degree of freedom for $\boldsymbol{W}$ as $n(\frac{kd}{n^2}+n^2) = \frac{kd}{n}+n^3$. With the mild condition that $kd > n^4$, then $\frac{kd}{n}$ dominates and the overall parameter size of the PHM layer in (5.4) is $\mathcal{O}(\frac{kd}{n})$. This condition is satisfied for typical values for adapters, PHM layers, and large-scale PLMs such as T5-large, with hidden size $k=1024$, adapter hidden size $d\in\{24,32,48,96\}$, and $n=2,4,8,12$. Hence, the PHM layer offers a parameter reduction of almost $\frac{1}{n}$ compared to standard fully-connected layers, which are $\mathcal{O}(kd)$.[4]

Similarly, employing PHM layers for modeling down and up-projection matrices offers a parameter reduction of almost $\frac{1}{n}$. Each adapter with a PHM layer has in total $2(\frac{kd}{n}+n^3)$ parameters. For a Transformer model with L layers, the total number of parameters of PHM-ADAPTER is $4L(\frac{kd}{n}+n^3)$.

COMPACTER parameters COMPACTER shares the trained weight matrices $\{\boldsymbol{A_i}\}_{i=1}^n$ in (5.5) consisting of n^3 parameters across all layers. COMPACTER also has two rank-one weights for each adapter, $\boldsymbol{s_i}$,$\boldsymbol{t_i}$ in (5.5) consisting of $\frac{k}{n}+\frac{d}{n}$ parameters, resulting in a total of $2n(\frac{k}{n}+\frac{d}{n})$ parameters for down and up-projection weights. Therefore, the total number of parameters of COMPACTER is $4L(k+d)+n^3$ for a transformer with L layers in the encoder and decoder.

In settings with a large number of layers, the dominant term is $4L(k+d)$. Therefore, with a mild condition that $4L(k+d)>n^3$, COMPACTER has a complexity of $\mathcal{O}(k+d)$, which is far more efficient compared to adapters' $\mathcal{O}(kd)$ and PHM-ADAPTER's $\mathcal{O}(\frac{kd}{n})$ complexity respectively. In settings where n is large, the number of parameters for shared weight matrices $\{\boldsymbol{A_i}\}_{i=1}^n$ for all layers remain constant in COMPACTER with a total of n^3 parameters while this scales linearly with the number of layers L for PHM and adapter layers. As an example, in the T5_{BASE} model with 222M parameters (Raffel et al., 2020), COMPACTER only learns 0.047% of the parameters, and maintains comparable performance to *full fine-tuning*.

5.5 Experiments

Datasets Following Raffel et al. (2020), we evaluate the performance of the methods on the GLUE (Wang et al., 2019c) and SUPERGLUE (Wang et al., 2019b) benchmarks. These benchmarks cover multiple tasks of paraphrase detection (MRPC, QQP), sentiment classification (SST-2), natural language inference (MNLI, RTE, QNLI, CB), linguistic acceptability (CoLA), question-answering

[4]Even for smaller models where the n^4 term dominates, we observe a substantial reduction of parameters compared to adapters.

(MultiRC, ReCoRD, BoolQ), word sense disambiguation (WiC), and sentence completion (COPA).[5] As the original test sets are not publicly available, we follow Zhang et al. (2021b) and split off 1k samples from the training set that we use for validation, while we use the original validation data as the test set. For datasets with fewer than 10k samples (RTE, MRPC, STS-B, CoLA, COPA, WiC, CB, BoolQ, MultiRC), we divide the original validation set in half, using one half for validation and the other for testing.

Experimental details We use the state-of-the-art encoder-decoder T5 model (Raffel et al., 2020) as the underlying model for all methods in our experiments. For computational efficiency, we report all results on T5_{BASE} models (12 encoder and decoder layers and 222M parameters). We use its HuggingFace PyTorch implementation (Wolf et al., 2020). We fine-tune all methods for 3 epochs on large datasets and 20 epochs for low-resource datasets of GLUE (MRPC, CoLA, STS-B, RTE, BoolQ, CB, COPA, WiC) to allow the models to converge (Zhang et al., 2021b). For all adapter-based methods, we experiment with adapters of bottleneck size of $\{96,48,24\}$. We save a checkpoint every epoch for all models and report the results for the hyper-parameters performing the best on the validation set for each task. For the PHM layers, we use the PyTorch implementation of Le et al. (2021). We include low-level details in Appendix 5.8. For our methods, we experiment with $n=\{4,8,12\}$ and report the model performing the best. We include the results for all values of n in Appendix 5.9.

Following Mahabadi et al. (2021b), we freeze the output layer of the pretrained model for all tasks across all methods.[6] We show the results with fine-tuning the output layer in Appendix 5.10. Following Houlsby et al. (2019), we update the layer normalization parameters for all methods where applicable.[7]

5.5.1 Baselines

We compare against several recently proposed *parameter-efficient* fine-tuning methods:

T5_{BASE} We compare our method to the standard practice of fine-tuning T5, where we fine-tune all parameters of the model on each individual task.

Adapters We compare to a strong adapter baseline (Houlsby et al., 2019), which adds adapters for each task after the feed-forward and attention modules in each transformer block of T5.

PFEIFFER-ADAPTER Pfeiffer et al. (2021) propose a more efficient adapter variant, which keeps only one of the adapters in each layer for better training efficiency. We experimented with keeping either adapter and found keeping the adapter after the self-attention module in each layer to perform the best.

ADAPTER-LOWRANK We parameterize each adapter's weight as a product of two rank-one weights.

[5] Following Raffel et al. (2020); Devlin et al. (2019), as a common practice, we do not experiment with WNLI (Levesque et al., 2012) due to its adversarial nature with respect to the training set.

[6] This is much more efficient as the output layer includes 11.1% of the parameters of T5_{BASE}. Tasks are formulated in a text-to-text format so the model can be applied to them without learning a new output layer (Raffel et al., 2020). We note that this is in contrast to the original adapter setting, which used an encoder-only masked PLM (Houlsby et al., 2019).

[7] For BITFIT, we only update the biases. For PROMPT TUNING, the entire model is frozen.

PROMPT TUNING Prompt tuning (Lester et al., 2021) is the successor variant of Li and Liang (2021), which prepends a randomly initialized continuous prompt to the input (PROMPT TUNING-R). We also compare to a variant, which initializes prompts using token embeddings of the pretrained language model's vocabulary (PROMPT TUNING-T) (Lester et al., 2021).

INTRINSIC-SAID The Structure Aware Intrinsic Dimension (Aghajanyan et al., 2021) fine-tunes the model by reparameterizing the parameters in a lower-dimensional subspace $\boldsymbol{\theta}^{\boldsymbol{d'}}$ ($d' \ll D$): $\boldsymbol{\theta}_{\boldsymbol{i}}^{\boldsymbol{D}} = \boldsymbol{\theta}_{\boldsymbol{i,0}}^{\boldsymbol{D}} + \lambda_i \boldsymbol{P} \boldsymbol{\theta}_{\boldsymbol{i}}^{\boldsymbol{d'-m}}$ where parameter $\boldsymbol{\theta}_{\boldsymbol{i,0}}^{\boldsymbol{D}}$ are the pretrained model's parameters and $\boldsymbol{P} \in \mathbb{R}^{d'-m} \rightarrow \mathbb{R}^D$ is a random linear projection via the Fastfood transform (Le et al., 2013). They then consider the total number of weight matrices in the PLM, m, and attribute a weight to each of them, resulting in $\boldsymbol{\lambda} \in \mathbb{R}^m$ in total by trading m parameters from the low dimensional space $\boldsymbol{\theta}^{\boldsymbol{d'}} \in \mathbb{R}^{d'}$. Then, the total trainable parameters are $\boldsymbol{\theta}^{\boldsymbol{d'-m}} \in \mathbb{R}^{d'-m}$ and $\boldsymbol{\lambda}$.

ADAPTERDROP We apply the method of Rücklé et al. (2021), which drops the adapters from lower transformer layers for a better training efficiency to T5 with Adapters. Consequently, we drop adapters from the first five layers of both the encoder and the decoder in T5_{BASE}.

BITFIT Cai et al. (2020) propose to freeze the weights and only train the biases. By not storing intermediate activations, this method enables substantial memory savings. Ravfogel et al. (2021) study a similar method for PLMs that fine-tunes only the biases and the final output layer.[8]

5.5.2 Our Methods

PHM-ADAPTER We learn the weights of adapters using PHM layers as in (5.4). To our knowledge, we are the first who exploit the idea of PHM (Zhang et al., 2021a) for efficient *fine-tuning* of large-scale language models.

COMPACTER We learn adapter weights using LPHM layers as described in (5.5). We also explore a variant where we only keep the COMPACTER layer after the feed-forward layer in each transformer block (COMPACTER++).[9]

5.5.3 Results on the GLUE Benchmark

Table 5.1 shows the results on GLUE with T5_{BASE} (see Appendix 5.12 for results on T5_{SMALL}). COMPACTER and COMPACTER++ outperform all previous parameter-efficient methods and perform on par with full fine-tuning while only training 0.07% and 0.047% of parameters respectively. We now discuss the different methods in detail.

Adapter-based methods For ADAPTER, not fine-tuning the classifier hurts the performance substantially (85.78 versus 86.48; cf. Appendix 5.10). PFEIFFER-ADAPTER, which adds adapters only after the

[8] Note that in the HuggingFace T5 implementation, the biases in layer normalizations, linear layers, the output layer and self-attention layers are removed. We re-introduce these biases for BITFIT.

[9] We found this to slightly outperform keeping the COMPACTER layer after the self-attention layer instead.

Table 5.1: Performance of all models on the GLUE tasks. For each method, we report the total number of parameters across all tasks and the number of parameters that are trained for each task as a multiple and proportion of $T5_{BASE}$ model (Raffel et al., 2020). For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy. Bold fonts indicate the best results. For the results with †, due to insatiability during training, we restarted experiments with 6 random seeds and report the best. For INTRINSIC-SAID, d' is set to 20K.

Method	#Total params	Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
Baselines											
$T5_{BASE}$	8.0×1	100%	61.76	**94.61**	**90.20/93.06**	**91.63/88.84**	89.68/89.97	**86.78**	93.01	71.94	**86.50**
ADAPTER	1.065	0.832%	**64.02**	93.81	85.29/89.73	90.18/87.20	90.73/91.02	86.49	93.21	71.94	85.78
PFEIFFER-ADAPTER	1.032	0.427%	62.9	93.46	86.76/90.85	90.14/87.15	91.13/91.34	86.26	93.30	**76.26**	86.32
ADAPTERDROP	1.038	0.494%	62.7	93.58	86.27/90.60	90.2/87.25	**91.37/91.61**	86.27	93.23	71.22	85.85
ADAPTER-LOWRANK	1.004	0.073%	59.19	93.69	88.24/91.49	90.23/87.01	90.8/91.33	85.8	92.9	73.38	85.82
PROMPT TUNING-R	1.003	0.034%	0.47†	87.61	68.14/81.05	88.93/85.55	90.25/90.59	46.83†	92.33	54.68	71.49
PROMPT TUNING-T	1.003	0.034%	10.59	90.94	68.14/81.05	89.69/86.14	89.84/90.21	81.46	92.75	54.68	75.95
INTRINSIC-SAID	1.001	0.009%	58.69	94.15	88.24/91.78	90.28/87.13	90.06/90.45	85.23	**93.39**	70.50	85.45
BITFIT	1.010	0.126%	58.16	94.15	86.76/90.53	90.06/86.99	90.88/91.26	85.31	92.99	67.63	84.97
Our Proposed Methods											
PHM-ADAPTER ($n=12$)	1.013	0.179%	57.35	**94.50**	**91.67/93.86**	**90.25/87.05**	90.45/90.84	**85.97**	92.92	75.54	86.40
COMPACTER ($n=4$)	1.004	0.073%	**63.75**	93.00	89.22/92.31	90.23/87.03	90.31/90.74	85.61	92.88	**77.70**	**86.62**
COMPACTER++ ($n=4$)	1.002	0.047%	61.27	93.81	90.69/93.33	90.17/86.93	**90.46/90.93**	85.71	**93.08**	74.82	86.47

self-attention module outperforms the standard Adapters while being more parameter-efficient. ADAPTERDROP obtains lower performance than fine-tuning, demonstrating that adapting the lower layers of an encoder-decoder T5 model is important for its performance. Additionally, ADAPTER-LOWRANK is not expressive enough to perform well on this benchmark.

Prompt tuning and BitFit For PROMPT TUNING, we observe high sensitivity to initialization and learning rate, as also confirmed in Li and Liang (2021). We experimented with multiple random seeds but performance lags behind fine-tuning substantially, in particular on low-resource datasets. This can be explained by the low flexibility of such methods as all the information needs to be contained in the prefixes. As a result, the method only allows limited interaction with the rest of the model and good performance requires very large models (Lester et al., 2021). In addition, increasing the sequence length leads to memory overhead (see §5.5.5) and the number of prompt tokens is limited by the number of tokens that can fit in the model's maximum input length, which makes such methods less flexible and unsuitable for dealing with large contexts. Similarly, BITFIT performs worse than fine-tuning, especially on low-resource datasets.

Intrinsic-SAID Interestingly, the average performance of INTRINSIC-SAID, which fine-tunes only 0.009% of a model's parameters is only 1.05 points below the fine-tuning baseline. However, this method has two practical drawbacks: a) storing the random projection matrices results in a substantial memory overhead; b) it is very slow to train (see §5.5.5). Despite this, INTRINSIC-SAID provides in-

sights regarding the effectiveness of low-rank optimization of pretrained language models (Aghajanyan et al., 2021), which motivates the development of parameter-efficient methods such as COMPACTER.

COMPACTER For our proposed methods, we observe fine-tuning the output layer for both PHM-ADAPTER and COMPACTER++ does not provide much performance difference (see Appendix 5.10). PHM-ADAPTER reduces the parameters of ADAPTER from 0.83% to 0.179% (with $n=12$), being 4.64$\times$ more parameter-efficient. COMPACTER reduces the number of parameters to the remarkable rate of 0.073% while obtaining comparable results to full fine-tuning. By removing the COMPACTER layer after self-attention, COMPACTER++ obtains similar performance, while reducing the parameters to 0.047%. Adaptation without updating the layer normalization can be a promising direction to reduce the parameters further, for instance by building on recent advances in normalization-free models (Brock et al., 2021), which we leave to future work.

5.5.4 Results on the SUPERGLUE Benchmark

Table 5.2 shows the performance of the methods on SUPERGLUE (Wang et al., 2019b). We include the results for all values of n in Appendix 5.11. We observe a similar pattern as on GLUE in Table 5.1. COMPACTER and COMPACTER++ perform substantially better compared to other parameter-efficient fine-tuning methods and even outperform full fine-tuning while only training 0.073% and 0.048% of the parameters.

5.5.5 Efficiency Evaluation

In this section, we compare the efficiency of our proposed methods with various recently proposed parameter-compact fine-tuning methods under the same computation budget. To this end, we train all methods for 1 epoch on the MNLI dataset. For each method, we select the largest batch size that fits a fixed budget of the GPU memory (24 GB). For all adapter-based methods, we fix the adapter size to 24. For PROMPT TUNING, we set the number of prefix tokens to 100. For INTRINSIC-SAID, we set $d'=1400$. Finally, we set $n=4$. In Table 5.3, we report the percentage of trained parameters per task, training time per epoch, and memory usage of each method. Moreover, Figure 5.1 shows the trade-off between quantitative performance, percentage of trained parameters, and memory footprint.

Our approaches have several attractive properties. Based on our analysis in Table 5.1, COMPACTER and COMPACTER++ obtain the best combination of high GLUE score averaged across all tasks, plus a substantially lower number of parameters (0.073% and 0.047% respectively). In addition to COMPACTER++ performing well, its memory requirement is the second best among all methods, reducing memory usage by -41.94% compared to T5_{BASE}. COMPACTER and COMPACTER++ also speed up training substantially, by -13.41% and -26.51% relative to T5_{BASE}. On the other hand, BITFIT, by not storing intermediate activations, has the lowest memory requirement (-64.2% relative to T5_{BASE}) and is the fastest (-35.06% relative to T5_{BASE}) at the cost of lower quantitative performance (1.53 points lower; see Table 5.1).

Table 5.2: Performance of all methods on the SUPERGLUE tasks. For each method, we report the total number of parameters across all tasks and the percentage of parameters that are trained for each task as a multiple and proportion of $T5_{BASE}$ model (Raffel et al., 2020). For CB, we report accuracy and F1. For MultiRC, we report F1 over all answer-options ($F1_a$) and exact match of each question's set of answers (EM) Wang et al. (2019b). For ReCoRD, we report F1 and EM scores. For all other tasks, we report accuracy. For INTRINSIC-SAID, d' is set to 20K. Bold fonts indicate the best results in each block.

Method	#Total params	Trained params / per task	BoolQ	CB	COPA	MultiRC	ReCoRD	WiC	Avg
Baselines									
$T5_{BASE}$	6.0×1	100%	81.10	**85.71/78.21**	52.0	68.71/47.0	74.26/73.33	**70.22**	70.06
Adapters	1.049	0.832%	82.39	85.71/73.52	52.0	72.75/53.41	74.55/73.58	67.08	70.55
PFEIFFER-ADAPTER	1.024	0.427%	**82.45**	85.71/75.63	54.0	72.53/51.76	74.69/73.70	68.65	**71.01**
ADAPTERDROP	1.028	0.494%	82.26	85.71/75.63	42.0	**72.92/53.30**	74.68/73.70	68.34	69.84
ADAPTER-LOWRANK	1.003	0.073%	80.31	78.57/55.37	54.0	72.58/51.98	74.77/73.87	64.58	67.34
PROMPT TUNING-R	1.002	0.034%	61.71	67.86/46.99	48.0	59.23/16.33	75.27/74.36	48.90	55.41
PROMPT TUNING-T	1.002	0.034%	61.71	67.86/46.89	52.0	57.66/19.44	**75.37/74.41**	48.90	56.03
INTRINSIC-SAID	1.001	0.009%	78.72	75.00/51.83	54.0	69.98/52.78	74.86/73.91	65.83	66.32
BITFIT	1.008	0.126%	79.57	78.57/54.40	**56.0**	70.73/48.57	74.64/73.64	69.59	67.30
Our Proposed Methods									
PHM-ADAPTER ($n=4$)	1.013	0.240%	**80.31**	85.71/73.52	44.0	**71.99/51.65**	**74.62/73.60**	67.40	69.20
COMPACTER ($n=12$)	1.003	0.073%	78.59	**96.43/87.44**	48.0	70.80/49.67	74.49/73.54	65.20	71.57
COMPACTER++ ($n=12$)	1.002	0.048%	78.84	92.86/84.96	**52.0**	70.68/50.99	74.55/73.50	**68.03**	**71.82**

Table 5.3: Percentage of trained parameters per task, average peak memory and training time for all methods. $\Delta\%$ is the relative difference with respect to *full fine-tuning* ($T5_{BASE}$). Lower is better.

Method	Trained params/ per task	Memory (MB)	$\Delta\%$	Time/ Epoch (min)	$\Delta\%$
$T5_{BASE}$	100%	167.99	—	42.13	—
ADAPTER	0.832%	124.02	-35.45%	31.81	-24.50%
PFEIFFER-ADAPTER	0.427%	118.4	-41.88%	28.19	-33.09%
ADAPTERDROP	0.494%	119.41	-40.68%	28.08	-33.35%
ADAPTER-LOWRANK	0.073%	123.8	-35.69%	32.71	-22.36%
PROMPT TUNING	0.034%	222.27	24.42%	44.54	5.72%
INTRINSIC-SAID	0.009%	285.40	41.14%	144.01	241.82%
BITFIT	0.126%	102.31	-64.20%	27.36	-35.06%
PHM-ADAPTER	0.179%	123.93	-35.55%	35.55	-15.62%
COMPACTER	0.073%	123.91	-35.57%	36.48	-13.41%
COMPACTER++	0.047%	118.35	-41.94%	30.96	-26.51%

Methods relying on pruning adapters, i.e., PFEIFFER-ADAPTER and ADAPTERDROP reduce the memory overhead and improve training time. However, their number of parameters is almost an order of magnitude more compared to COMPACTER++, with 9.1× and 10.5× more parameters respectively. Moreover, although, PFEIFFER-ADAPTER performs on par with full fine-tuning with a slight degradation (Table 5.1), ADAPTERDROP obtains a lower performance (-0.65 less on average across all tasks.). We note that dropping adapters from transformer layers is a general technique and could be applied to COMPACTER for improving efficiency even further, which we leave to future work. Similarly, although ADAPTER-LOWRANK reduces the memory overhead and improves the training time, it obtains a lower performance (Table 5.1) (-0.68 less on average across all tasks.).

At the other end of the spectrum, INTRINSIC-SAID and PROMPT TUNING methods have the lowest number of parameters. However, they both come with high memory overhead (41.14% and 24.42% relative to full fine-tuning ($T5_{BASE}$) respectively), are slowest to train, and their performance substantially lags behind full fine-tuning (see Table 5.1). For PROMPT TUNING, high memory costs are due to the fact that the computational complexity of self-attention, which requires storing the full attention matrix for gradient computation, scales quadratically with the sequence length Wang et al. (2020). For INTRINSIC-SAID, the high memory requirement is due to storing large random projection matrices, which limits the application of INTRINSIC-SAID for fine-tuning large-scale PLMs. Moreover, computing projections via FastFood transform, although theoretically possible in $O(D\log d')$ (Le et al., 2013), is slow in practice even with a CUDA implementation. For pretrained language models with a large number of parameters, allocating random projections for the full parameter space is intractable. While using Fastfood transform partially ameliorates this issue by reducing the memory usage from $\mathcal{O}(Dd')$ to $\mathcal{O}(D)$, the memory issue with such methods remains unresolved.

Overall, given the size of large-scale transformer models with millions and billions of parameters, such as T5 (Raffel et al., 2020), efficient memory usage is of paramount importance for practical applications. COMPACTER and COMPACTER++ offer a great trade-off in terms of performance, memory usage, and training time. With regard to our inspiration of von Neumann's quotation, we thus find that only a comparatively small number of additional parameters are necessary for the practical and efficient adaptation of PLMs.

5.5.6 Low-resource Fine-tuning

COMPACTER++ has substantially fewer parameters compared to $T5_{BASE}$. In this section, we investigate whether this could help COMPACTER++ to generalize better in resource-limited settings. We subsample each dataset of GLUE for varying sizes in the range $\{100,500,1000,2000,4000\}$. Figure 5.4 shows the results. COMPACTER++ substantially improves the results in the low-resource setting, indicating more effective fine-tuning in this regime.

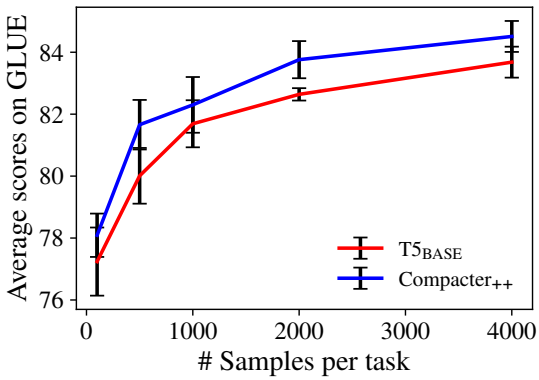

Figure 5.4: Results on GLUE for the various number of training samples per task (100,500,1000,2000,4000). We show mean and standard deviation across 5 seeds.

5.6 Related Work

Adapters Adapters have recently emerged as a new paradigm for fine-tuning pretrained language models (Houlsby et al., 2019). In another line of work, Üstün et al. (2020) proposed a multilingual dependency parsing method based on adapters and contextual parameter generator networks (Platanios et al., 2018), where they generate adapter parameters conditioned on trained input language embeddings. This, however, leads to a large number of additional parameters compared to the base model. Contemporaneously, Mahabadi et al. (2021b) use a single compact hypernetwork allowing to generate adapter weights efficiently conditioned on multiple tasks and layers of a transformer model. Pilault et al. (2021) also proposed a task-conditioned transformer for multi-task learning which is less parameter-efficient. The aforementioned work is complementary to COMPACTER, and one could potentially combine COMPACTER with contextual parameter generation to generate adapter modules. Compared to Mahabadi et al. (2021b), COMPACTER++ reduces the parameters by 6.2×.

Hypercomplex representations Deep learning advances in the hypercomplex domain are in a nascent stage, and most work is fairly recent (Gaudet and Maida, 2018; Parcollet et al., 2018b,a; Zhu et al., 2018; Tay et al., 2019). Replacing matrix multiplications in standard networks with Hamilton products that have fewer degrees of freedom offers up to a 4× saving of parameter size in a single multiplication operation (Parcollet et al., 2018a; Tay et al., 2019). Very recently, Zhang et al. (2021a) extend such methods in a way that they could reduce the parameters of a fully connected layer under a mild condition to $1/n$, where n is a user-specified parameter. To the best of our knowledge, there is no previous work that attempts to leverage the hypercomplex space for efficient fine-tuning of large-scale language models.

Other parameter-efficient models Li et al. (2018) and Aghajanyan et al. (2021) study training models in a low-dimensional randomly oriented subspace instead of their original parameter space. Another recent line of work has shown that pretrained models such as BERT are redundant in their capacity, allowing for significant sparsification without much degradation in end metrics (Chen et al., 2020; Prasanna et al., 2020; Desai et al., 2019). Such methods, however, remain not well supported by current hardware and often perform worse compared to dedicated efficient architectures (Blalock et al., 2020).

5.7 Conclusion

We have proposed COMPACTER, a light-weight fine-tuning method for large-scale language models. COMPACTER generates weights by summing Kronecker products between shared "slow" weights and "fast" rank-one matrices, specific to each COMPACTER layer. Leveraging this formulation, COMPACTER reduces the number of parameters in adapters substantially from $\mathcal{O}(kd)$ to $\mathcal{O}(k+d)$. Through extensive experiments, we demonstrate that despite learning 2127.66$\times$ fewer parameters than standard fine-tuning, COMPACTER obtains comparable or better performance in a full-data setting and outperforms fine-tuning in data-limited scenarios.

Appendix

5.8 Experimental Details

Datasets We run all experiments on the standard GLUE benchmark (Wang et al., 2019c) with Creative Commons license (CC BY 4.0) and the SUPERGLUE benchmark Wang et al. (2019b). These benchmark consist of multiple datasets: CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), QQP[10], STS-B (Cer et al., 2017), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), and RTE, which is a combination of data from RTE1 (Dagan et al., 2005), RTE2 (Bar-Haim et al., 2006), RTE3 (Giampiccolo et al., 2007), RTE5 (Bentivogli et al., 2009), COPA (Roemmele et al., 2011), CB (De Marneffe et al., 2019), MultiRC (Khashabi et al., 2018), ReCoRD (Zhang et al., 2018), BoolQ (Clark et al., 2019a), and WiC (Pilehvar and Camacho-Collados, 2019) where sentences are selected from VerbNet (Schuler, 2005), WordNet (Miller, 1995), and Wiktionary. We download all datasets from the HuggingFace Datasets library (Lhoest et al., 2021).

Low-resource fine-tuning For the experiment conducted in §5.5.6, we set the number of epochs to 1000, 200, 100, 50, 25, for datasets subsampled to size 100, 500, 1000, 2000, and 4000 respectively. Based on our results, this is sufficient to allow the models to converge. We save a checkpoint every 250 steps for all models and report the results for the hyper-parameters performing the best on the validation set for each task.

Data pre-processing Following Raffel et al. (2020), we cast all datasets into a sequence-to-sequence format. We recast STS-B as a 21-class classification task by rounding its target scores to their nearest increment of 0.2.

Computing infrastructure We run the experiments in Table 5.1, 5.2, 5.9, and 5.3 on one NVIDIA GEFORCE RTX 3090, and experiments in §5.5.6 on one GEFORCE GTX 1080 TI GPU.

Training hyper-parameters For the experiments on GLUE, we set the maximum sequence length to 128 and batch size to 100. Following Raffel et al. (2020), we use maximum sequence length of

[10] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

Table 5.4: Selected learning rates for all methods.

Method	Learning rate
$T5_{BASE}$	3e−4
ADAPTER	3e−4
PFEIFFER-ADAPTER	3e−4
ADAPTERDROP	3e−4
ADAPTER-LOWRANK	3e−3
PROMPT TUNING-R	3e−2
PROMPT TUNING-T	3e−1
INTRINSIC-SAID	3e−2
BITFIT	3e−4
PHM-ADAPTER	3e−3
COMPACTER	3e−3
COMPACTER++	3e−3

Table 5.5: Selected learning rates for all methods, when we also fine-tune the output layer.

Method	Learning rate
Adapters	3e−3
PFEIFFER-ADAPTER	3e−4
ADAPTERDROP	3e−4
ADAPTER-LOWRANK	3e−3
BITFIT	3e−4
PHM-ADAPTER	3e−3
COMPACTER	3e−3
COMPACTER++	3e−3

256 for the tasks in SUPERGLUE, and for ReCoRD, we set it to 512. We used batch size of 32 for SUPERGLUE, and for ReCoRD, we set it to 16 due to the GPU memory limit. For results in §5.5.6, we set the batch size to 40 to match the lower GPU memory of GEFORCE GTX 1080 TI GPU. For setting the learning rates, we trained all methods with 3e−5, 3e−4, 3e−3, 3e−2, and 3e−1 and use the learning rate performing the best on the validation set for each method. Table 5.4 shows the final selected learning rate for each method reported in Table 5.1. For the method variants where we also fine-tune the final output layer (Table 5.7), we report the selected learning rate in Table 5.5. We train all models with the AdamW optimizer from the HuggingFace library (Wolf et al., 2020) with default hyper-parameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e{-}8$. We set warm-up steps to 500 for all methods in Table 5.1 and 5.7. We set the warm-up steps to 0 for all methods in Table 5.2 and 5.9, which based on our experiments, improved the results for all methods.

5.9 Impact of Hyper-parameters

In this section, we study the impact of hyper-parameters for each method reported in Table 5.1. We report the results in Table 5.6.

Impact of dimension (d') on INTRINSIC-SAID Increasing the dimension d' for INTRINSIC-SAID method often improves results. Though, as discussed in (Aghajanyan et al., 2021), d' is task-dependent so needs to be tuned for every new dataset to achieve optimal performance.

Impact of n on PHM-ADAPTER Table 5.6 shows the results for varying values of $n = \{4,8,12\}$. We experiment with adapters of bottleneck size $d \in \{24,48,96\}$.

For the $T5_{BASE}$ model with $k = 768$, the condition $kd > n^4$ discussed in §5.4 is partially satisfied for $d = 24$ and $n = 4,8$ and fully satisfied for $d \in \{48,96\}$ and $n = 4,8,12$. Note that this condition is

satisfied for larger versions of the T5 model, i.e., T5-large (770 million parameters, $k=1024$), T5-3B (2.8 billion parameters, $k=1024$), and T5-11B (11 billion parameters, $k=1024$) with adapter hidden size $d \in \{24,32,48,96\}$ and $n=2,4,8,12$. Due to the huge computational costs of training these models, we could not run experiments on such a large scale. Nevertheless, we observe substantial parameter reduction using PHM-ADAPTER.

In Table 5.6, we report the number of parameters for $d=24$ for all methods. Compared to Adapters, PHM-ADAPTER with $n=8$ reduces the parameters substantially by 5.2×.

Impact of n on COMPACTER For COMPACTER and COMPACTER++, we observe that the number of trainable parameters is almost constant across different values of n. This is due to the fact that the number of trainable parameters in layernorms (LN) and biases (B) in each LPHM layer make up a high proportion of parameters for our methods. For instance for $n=4$, for COMPACTER with 0.073% of trainable parameters, LN and B make up 28.49% and 23.51% respectively of its trainable parameters; for COMPACTER++ with 0.047% of trainable parameters, LN and B make up 44.01% and 18.15% respectively of its parameters; while for PHM-ADAPTER with 0.239% of trainable parameters, LN and B make up only 8.63% and 7.12% respectively of its parameters. Consequently, simply removing biases from adapters, and exploring ideas of training language models without layer normalizations (Brock et al., 2021) can be promising directions on reducing parameters further, which we leave to future work.

COMPACTER has more than an order of magnitude fewer parameters compared to Adapters, with a parameter reduction at a remarkable rate of 11.4×. COMPACTER++ even reduces the parameters further by 17.7× in total.

5.10 Results with Fine-tuning the Output Layer

Table 5.7 shows the results for the methods in Table 5.1 with fine-tuning the output layer. The parameters of the output layer dominate the parameters of each method and thus reduce the relative parameter savings. The standard adapter obtains the largest improvement in performance when fine-tuning the output layer compared to the results in Table 5.1. In contrast, our proposed methods perform well with or without fine-tuning the output layer.

5.11 Results on SUPERGLUE

Table 5.8 shows the performance of our proposed methods on SUPERGLUE for different values of n. We include the learning rate obtaining the best validation performance for all methods reported in Table 5.2 in Table 5.11.

Table 5.6: Performance of all methods on the tasks in GLUE for different values of hyper-parameters. For each method, we report the total number of parameters across all tasks and the number of parameters that are trained for each task as a multiple and proportion of $T5_{BASE}$ model (Raffel et al., 2020). For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy. Bold fonts indicate the best results in each block.

Method	#Total params	Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
INTRINSIC-SAID ($d'=0.4$K)	1.001	0.0002%	0.0	92.55	78.43/85.62	90.25/87.19	90.43/90.66	69.93	89.31	58.99	75.76
INTRINSIC-SAID ($d'=1.4$K)	1.001	0.0006%	52.40	93.35	**89.22/92.41**	90.44/87.31	89.86/90.23	82.01	93.12	67.63	84.36
INTRINSIC-SAID ($d'=2.5$K)	1.001	0.0011%	45.78	93.92	89.22/92.20	**90.43/87.37**	90.32/90.90	82.86	93.17	64.03	83.65
INTRINSIC-SAID ($d'=10$K)	1.001	0.0045%	56.13	93.58	88.73/91.99	90.34/87.18	**90.63/90.99**	84.84	93.36	**71.22**	85.36
INTRINSIC-SAID ($d'=20$K)	1.001	0.0090%	**58.69**	**94.15**	88.24/91.78	90.28/87.13	90.06/90.45	**85.23**	**93.39**	70.50	**85.45**
PHM-ADAPTER ($n=4$)	1.018	0.239%	59.21	93.69	87.25/90.91	90.23/86.99	90.55/90.73	85.93	**93.04**	69.78	85.30
PHM-ADAPTER ($n=8$)	1.011	0.160%	**61.84**	93.58	91.18/93.57	**90.25/87.08**	**90.74/91.07**	85.74	92.93	70.50	86.23
PHM-ADAPTER ($n=12$)	1.013	0.179%	57.35	**94.50**	**91.67/93.86**	90.25/87.05	90.45/90.84	**85.97**	92.92	**75.54**	**86.40**
COMPACTER ($n=4$)	1.004	0.073%	**63.75**	93.00	89.22/92.31	**90.23/87.03**	90.31/90.74	85.61	92.88	**77.70**	**86.62**
COMPACTER ($n=8$)	1.004	0.073%	61.78	**93.81**	90.20/93.10	**90.23/87.03**	90.16/90.44	**85.78**	93.08	74.10	86.34
COMPACTER ($n=12$)	1.004	0.073%	61.38	93.69	**91.18/93.71**	90.11/86.88	**90.53/90.98**	85.76	**93.12**	70.50	86.17
COMPACTER++ ($n=4$)	1.002	0.047%	61.27	93.81	90.69/93.33	90.17/86.93	**90.46/90.93**	**85.71**	**93.08**	**74.82**	**86.47**
COMPACTER++ ($n=8$)	1.002	0.047%	62.79	92.55	88.24/91.95	90.16/86.94	90.43/90.78	85.36	92.82	73.38	85.95
COMPACTER++ ($n=12$)	1.002	0.048%	**63.01**	**93.92**	**91.18/93.75**	**90.23/87.01**	90.40/90.65	85.46	92.88	71.22	86.34

5.12 Impact of Model Size

Table 5.9 shows the results of methods using $T5_{SMALL}$ (60M parameters) on GLUE. For all adapter-based methods, we experiment with adapters of bottleneck size of $\{16,32,64\}$. For our methods, we experiment with $n=\{4,8,16\}$.

All parameter-efficient fine-tuning methods are performing worse than full fine-tuning with this small model size. This is in contrast to the results of Table 5.1 and 5.2, where some parameter-efficient fine-tuning methods were able to perform on par or outperform full fine-tuning with the larger model size of $T5_{BASE}$ (222M parameters). Among all methods, adapters, and our proposed methods perform the best. We report the learning rate performing the best on the validation set of each method in Table 5.10.

Table 5.7: Performance of all methods on the tasks in GLUE, where the output layer is tuned. For each method, we report the total number of parameters across all tasks and the percentage of parameters that are trained for each task as a multiple and proportion of $T5_{BASE}$ model (Raffel et al., 2020). For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy. Bold fonts indicate the best results in each block.

Method	#Total params	Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
Baselines											
Adapters	1.065	11.89%	61.80	**94.15**	88.24/91.67	**90.27/87.05**	**91.51/91.71**	86.02	92.64	**76.26**	**86.48**
PFEIFFER-ADAPTER	1.032	11.49%	**64.76**	93.58	87.75/91.58	90.16/87.17	91.21/91.50	86.16	**93.30**	73.38	86.41
ADAPTERDROP	1.038	11.56%	61.67	93.69	84.80/89.20	90.14/87.17	90.92/91.34	**86.24**	93.23	73.38	85.62
ADAPTER-LOWRANK	1.004	11.13%	62.82	93.81	88.73/91.99	90.34/87.19	90.51/90.58	85.81	92.93	74.82	86.32
BITFIT	1.010	11.19%	57.13	**94.15**	**89.71/92.78**	90.07/87.02	90.91/91.22	85.34	93.06	68.35	85.43
Our Proposed Methods											
PHM-ADAPTER ($n=4$)	1.017	11.30%	62.79	93.58	**89.22/92.41**	90.23/87.01	90.61/90.81	**86.06**	92.95	**75.54**	**86.47**
PHM-ADAPTER ($n=8$)	1.011	11.22%	61.24	**94.38**	88.73/91.99	90.28/87.08	90.53/90.98	85.94	**93.03**	73.38	86.14
PHM-ADAPTER ($n=12$)	1.013	11.24%	**65.25**	93.69	88.73/92.04	**90.34/87.16**	**90.75/90.89**	85.74	92.92	72.66	86.38
COMPACTER ($n=4$)	1.004	11.13%	**61.27**	93.58	88.24/91.67	90.25/87.08	**90.67/91.02**	**85.82**	92.92	73.38	85.99
COMPACTER ($n=8$)	1.004	11.13%	60.31	**93.81**	89.71/92.63	90.23/87.02	90.49/90.85	85.19	**93.08**	71.94	85.93
COMPACTER ($n=12$)	1.004	11.13%	59.25	93.12	**91.18/93.75**	**90.31/87.16**	90.37/90.82	85.33	92.97	**75.54**	**86.35**
COMPACTER++ ($n=4$)	1.002	11.11%	**64.28**	**94.27**	90.20/92.96	**90.23/87.04**	**90.27/90.61**	**85.80**	**92.97**	73.38	**86.55**
COMPACTER++ ($n=8$)	1.002	11.11%	63.78	93.58	**90.20/93.01**	90.19/87.02	90.12/90.56	85.57	92.84	70.50	86.12
COMPACTER++ ($n=12$)	1.002	11.11%	62.05	93.23	87.75/91.58	90.19/86.97	90.08/90.48	85.52	92.75	**79.86**	86.41

Table 5.8: Performance of our proposed methods on the tasks in SUPERGLUE for different values of n. For each method, we report the total number of parameters across all tasks and the percentage of parameters that are trained for each task as a multiple and proportion of $T5_{BASE}$ model (Raffel et al., 2020). For CB, we report accuracy and F1. For MultiRC, we report F1 over all answer-options ($F1_a$) and exact match of each question's set of answers (EM) Wang et al. (2019b). For ReCoRD, we report F1 and EM scores. For all other tasks, we report accuracy. Bold fonts indicate the best results in each block.

Method	#Total params	Trained params / per task	BoolQ	CB	COPA	MultiRC	ReCoRD	WiC	Avg
PHM-ADAPTER ($n=4$)	1.013	0.24%	**80.31**	**85.71/73.52**	44.0	**71.99/51.65**	**74.62/73.60**	67.40	**69.20**
PHM-ADAPTER ($n=8$)	1.008	0.160%	79.39	82.14/69.87	44.0	71.49/50.77	74.46/73.48	67.71	68.15
PHM-ADAPTER ($n=12$)	1.009	0.179%	79.33	78.57/75.43	**52.0**	70.48/50.66	74.14/73.14	**68.65**	69.16
COMPACTER ($n=4$)	1.003	0.073%	**79.88**	89.29/82.51	42.0	**71.87/51.98**	**74.64/73.59**	65.83	70.18
COMPACTER ($n=8$)	1.003	0.073%	79.57	85.71/80.06	**56.0**	70.75/49.67	74.56/73.57	**70.85**	71.19
COMPACTER ($n=12$)	1.003	0.073%	78.59	**96.43/87.44**	48.0	70.80/49.67	74.49/73.54	65.20	**71.57**
COMPACTER++ ($n=4$)	1.002	0.047%	**79.94**	85.71/80.06	50.0	72.16/50.33	74.63/73.60	**68.34**	70.53
COMPACTER++ ($n=8$)	1.002	0.047%	78.23	82.14/70.87	48.0	**71.61/51.43**	**74.62/73.64**	67.71	68.69
COMPACTER++ ($n=12$)	1.002	0.048%	78.84	**92.86/84.96**	**52.0**	70.68/50.99	74.55/73.50	68.03	**71.82**

Table 5.9: Performance of all methods on the tasks in GLUE. For each method, we report the total number of parameters across all tasks and the percentage of parameters that are trained for each task as a multiple and proportion of $T5_{SMALL}$ model (Raffel et al., 2020). For MNLI, we report accuracy on the matched validation set. For MRPC and QQP, we report accuracy and F1. For STS-B, we report Pearson and Spearman correlation coefficients. For CoLA, we report Matthews correlation. For all other tasks, we report accuracy. Bold fonts indicate the best results in each block. We repeat the experiments marked with $*$ multiple times for different seeds, but they were not successful.

Method	#Total params	Trained params / per task	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	Avg
Baselines											
$T5_{SMALL}$	8×1	100%	**46.90**	**91.74**	87.25/90.97	**90.07/86.68**	88.75/89.20	**82.20**	**90.59**	**65.47**	**82.71**
Adapters	1.054	0.698%	36.88	90.83	**88.73/91.93**	88.09/84.06	88.98/89.34	80.50	89.75	62.59	81.06
ADAPTERDROP	1.009	0.139%	34.73	89.91	83.33/88.36	87.96/83.89	88.73/88.80	79.33	89.86	61.87	79.71
PFEIFFER-ADAPTER	1.027	0.363%	38.86	90.48	85.78/89.90	87.82/84.26	**89.24/89.56**	80.63	89.84	57.55	80.36
ADAPTER-LOWRANK	1.005	0.090%	40.55	90.60	84.80/89.20	88.01/83.98	88.04/88.27	79.92	89.95	61.15	80.41
PROMPT TUNING-R	1.007	0.085%	0.0*	86.35	68.14/81.05	87.48/83.91	87.35/87.87	76.27	88.49	50.36	72.48
PROMPT TUNING-T	1.007	0.085%	0.0*	79.59	71.08/82.18	87.76/83.55	87.48/87.76	74.65	89.02	57.55	72.78
BITFIT	1.015	0.190%	25.59	90.48	84.80/89.42	88.01/83.77	87.58/87.89	78.15	88.94	63.31	78.90
INTRINSIC-SAID	1.003	0.033%	0.0*	90.25	84.80/89.05	88.07/84.00	87.81/88.08	79.02	89.90	52.52	75.77
Our Proposed Methods											
PHM-ADAPTER ($n=4$)	1.015	0.216%	40.08	90.60	**86.27/90.21**	**88.26/84.25**	**89.56/89.88**	**80.73**	**90.10**	60.43	80.94
PHM-ADAPTER ($n=8$)	1.011	0.170%	37.85	90.48	82.84/87.72	88.08/84.07	89.07/89.46	80.68	89.64	61.87	80.16
PHM-ADAPTER ($n=16$)	1.031	0.414%	36.27	**90.83**	83.82/88.34	88.03/84.02	87.94/88.44	80.04	89.95	58.99	79.70
COMPACTER ($n=4$)	1.005	0.090%	**44.65**	89.45	84.80/89.20	88.00/83.96	88.19/88.47	79.54	89.66	64.03	80.90
COMPACTER ($n=8$)	1.005	0.091%	42.90	89.56	84.31/89.12	88.01/83.95	88.51/88.79	79.60	89.68	**66.19**	**80.97**
COMPACTER ($n=16$)	1.006	0.097%	40.12	89.22	85.29/89.86	88.08/84.06	89.28/89.60	79.87	89.71	59.71	80.44
COMPACTER++ ($n=4$)	1.003	0.059%	39.89	90.37	84.31/89.26	88.04/83.99	88.69/88.98	79.45	89.05	63.31	80.49
COMPACTER++ ($n=8$)	1.003	0.059%	34.98	90.37	83.82/88.50	88.02/83.99	88.87/89.30	79.39	89.57	64.03	80.08
COMPACTER++ ($n=16$)	1.003	0.065%	37.54	89.79	85.78/89.90	88.01/83.96	88.93/89.30	79.35	89.40	64.75	80.61

Table 5.10: Selected learning rates for all methods with $T5_{SMALL}$

Method	Learning rate
$T5_{SMALL}$	3e−4
Adapters	3e−3
PFEIFFER-ADAPTER	3e−4
ADAPTERDROP	3e−3
ADAPTER-LOWRANK	3e−3
PROMPT TUNING-R	3e−2
PROMPT TUNING-T	3e−1
INTRINSIC-SAID	3e−2
BITFIT	3e−3
PHM-ADAPTER	3e−3
COMPACTER	3e−3
COMPACTER++	3e−3

Table 5.11: Selected learning rates for all methods with $T5_{BASE}$ on SUPERGLUE.

Method	Learning rate
$T5_{BASE}$	3e−4
Adapters	3e−4
PFEIFFER-ADAPTER	3e−4
ADAPTERDROP	3e−4
ADAPTER-LOWRANK	3e−3
PROMPT TUNING-R	3e−2
PROMPT TUNING-T	3e−1
BITFIT	3e−4
INTRINSIC-SAID	3e−3
PHM-ADAPTER	3e−3
COMPACTER	3e−3
COMPACTER++	3e−3

6 PERFECT: Prompt-free and Efficient Few-shot Learning with Language Models

Current methods for few-shot fine-tuning of pretrained masked language models (PLMs) require carefully engineered prompts and verbalizers for each new task to convert examples into a cloze-format that the PLM can score. In this work, we propose PERFECT, a simple and efficient method for few-shot fine-tuning of PLMs *without relying on any such handcrafting*, which is highly effective given as few as 32 data points. PERFECT makes two key design choices: First, we show that manually engineered task prompts can be replaced with *task-specific adapters* that enable sample-efficient fine-tuning and reduce memory and storage costs by roughly factors of 5 and 100, respectively. Second, instead of using handcrafted verbalizers, we learn new *multi-token label embeddings* during fine-tuning, which are not tied to the model vocabulary and which allow us to avoid complex auto-regressive decoding. These embeddings are not only learnable from limited data but also enable nearly 100x faster training and inference. Experiments on a wide range of few shot NLP tasks demonstrate that PERFECT, while being simple and efficient, also outperforms existing state-of-the-art few-shot learning methods. Our code is publicly available at `https://github.com/facebookresearch/perfect.git`.

6.1 Introduction

Recent methods for few-shot language model tuning obtain impressive performance but require careful engineering of prompts and verbalizers to convert inputs to a cloze-format (Taylor, 1953) that can be scored with pre-trained language models (PLMs) (Radford et al., 2018, 2019; Brown et al., 2020; Schick and Schütze, 2021a,b). For example, as Figure 6.1 shows, a sentiment classifier can be designed by inserting the input text $\boldsymbol{x}$ in a *prompt template* "$\boldsymbol{x}$ It was [MASK]" where *verbalizers* (e.g., 'great' and 'terrible') are substituted for the [MASK] to score target task labels ('positive' or 'negative'). In this Chapter, we show that such engineering is not needed for few-shot learning and instead can be replaced with simple methods for data-efficient fine-tuning with as few as 32 end-task examples.

More specifically, we propose PERFECT, a Prompt-free and Efficient paRadigm for FEw-shot Cloze-based fine-Tuning. To remove handcrafted patterns, PERFECT uses *task-specific adapter layers* Houlsby et al. (2019); Pfeiffer et al. (2020) (§6.3.1). Freezing the underlying PLM with millions or

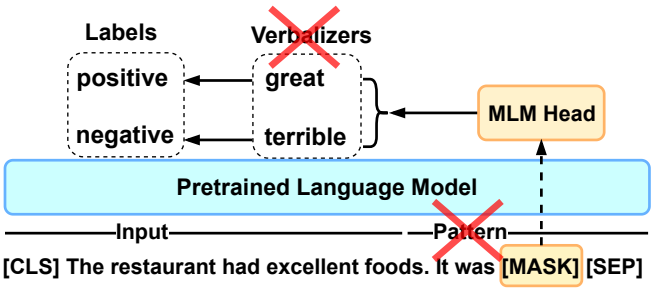

Figure 6.1: Existing few-shot fine-tuning methods require manual engineering to reduce new tasks to masked language modeling. PERFECT does not rely on any handcrafting, removing both patterns and verbalizers (see Figure 6.3).

billions of parameters (Liu et al., 2019b; Raffel et al., 2020), and only tuning adapters with very few new parameters saves on memory and storage costs (§6.4.2), while allowing very sample-efficient tuning (§6.4). It also stabilizes the training by increasing the worst-case performance and decreasing variance across the choice of examples in the few shot training sets (§6.4.3).

To remove handcrafted verbalizers (with variable token lengths), we introduce a new *multi-token fixed-length classifier scheme* that learns task label embeddings which are independent from the language model vocabulary during fine-tuning (§6.3.2). We show (§6.4) that this approach is sample efficient and outperforms carefully engineered verbalizers from *random initialization* (§6.4). It also allows us to avoid previously used expensive auto-regressive decoding schemes (Schick and Schütze, 2021b), by leveraging prototypical networks (Snell et al., 2017) over multiple tokens. Overall, these changes enable up to 100x faster learning and inference (§6.4.2).

PERFECT has several advantages: It avoids engineering patterns and verbalizers for each new task, which can be cumbersome. Recent work has shown that even some intentionally irrelevant or misleading prompts can perform as well as more interpretable ones Webson and Pavlick (2021). Unlike the zero-shot or extreme few-shot case, where prompting might be essential, we argue in this Chapter that all you need is tens of training examples to avoid these challenges by adopting PERFECT or a similar data-efficient learning method. Experiments on a wide variety of NLP tasks demonstrate that PERFECT outperforms state-of-the-art prompt-based methods while being significantly more efficient in inference and training time, storage, and memory usage (§6.4.2). To the best of our knowledge, we are the first to propose a few-shot learning method using the MLM objective in PLMs that provide state-of-the-art results while removing all per-task manual engineering.

6.2 Background

Problem formulation We consider a general problem of fine-tuning language models in a few-shot setting, on a small training set with K unique classes and N examples per class, such that the total number of examples is $|\mathcal{D}| = N \times K$. Let $\mathcal{D} = \cup_{k=1}^{K} \mathcal{D}_k$ be the given training set, where $\mathcal{D}_k = \{(\boldsymbol{x_k^i}, y_k^i)\}_{i=1}^{N}$ shows the set of examples labeled with class k and $y_k^i \in \mathcal{Y}$ is the corresponding label, where $|\mathcal{Y}| = K$.

We additionally assume access to a development set with the same size as the training data. Note that larger validation sets can grant a substantial advantage (Perez et al., 2021), and thus it is important to use a limited validation size to be in line with the goal of few-shot learning. Unless specified otherwise, in this work, we use 16 training examples ($N=16$) and a validation set with 16 examples, for a total of 32-shot learning.

6.2.1 Adapters

Recent work has shown that fine-tuning *all* parameters of PLMs with a large number of parameters in low-resource datasets can lead to a sub-optimal solution (Peters et al., 2019; Dodge et al., 2020). As shown in Figure 6.2, Rebuffi et al. (2018) and Houlsby et al. (2019) suggest an efficient alternative, by inserting small task-specific modules called *adapters* within layers of a PLMs. They then only train the newly added adapters and layer normalization, while fixing the remaining parameters of a PLM.

Each layer of a transformer model is composed of two primary modules: a) an attention block, and b) a feed-forward block, where both modules are followed by a skip connection. As depicted in Figure 6.2, adapters are normally inserted after each of these blocks before the skip connection.

Adapters are bottleneck architectures. By keeping input and output dimensions the same, they introduce no additional architectural changes. Each adapter, $A(.) \in \mathbb{R}^H$, consists of a down-projection, $D(.) \in \mathbb{R}^{H \times B}$, a non-linearity, such as GeLU (Hendrycks and Gimpel, 2016), and an up-projection $U(.) \in \mathbb{R}^{B \times H}$, where H is the dimension of input hidden states $\boldsymbol{x}$, and B is the bottleneck size. Formally defined as:

$$A(\boldsymbol{x}) = U(\text{GeLU}(D(\boldsymbol{x}))) + \boldsymbol{x}, \tag{6.1}$$

6.2.2 Prompt-based Fine-tuning

Standard Fine-tuning In standard fine-tuning with PLMs (Devlin et al., 2019), first a special [CLS] token is appended to the input $\boldsymbol{x}$, and then the PLM maps it to a sequence of hidden representations $\boldsymbol{h} = (\boldsymbol{h_1}, ..., \boldsymbol{h_S})$ with $\boldsymbol{h_i} \in \mathbb{R}^H$, where H is the hidden dimension, and S is the maximum sequence length. Then, a classifier, *softmax*($\boldsymbol{W^T h}_{[\text{CLS}]}$), using the embedding of the classification token ($\boldsymbol{h}_{[\text{CLS}]}$), is trained end-to-end for each downstream task. The main drawback of this approach is the discrepancy between the pre-training and fine-tuning phases since PLMs have been trained to *predict mask tokens* in a masked language modeling task (Devlin et al., 2019).

Prompt-based tuning To address this discrepancy, *prompt-based fine-tuning* (Schick and Schütze, 2021a,b; Gao et al., 2021) formulates tasks in a cloze-format (Taylor, 1953). This way, the model can predict targets with a *masked language modeling (MLM) objective*. For example, as shown in

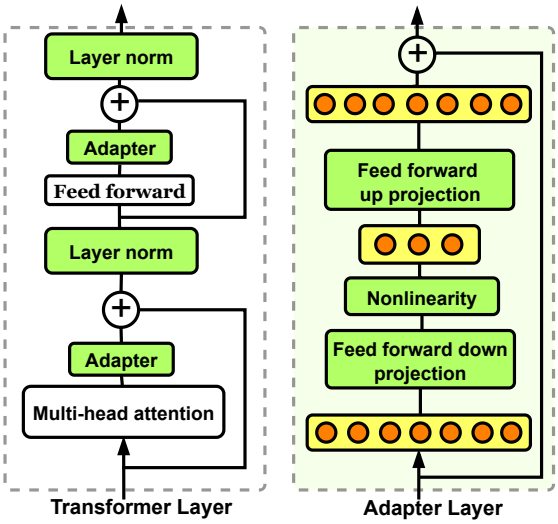

Figure 6.2: Left: Adapter integration in a PLM. Right: An adapter architecture. Adapters are usually inserted after the feed-forward and self-attention modules. During training, we only optimize the green components

Figure 6.1, for a sentiment classification task, inputs are converted to:

$$\boldsymbol{x}_{\text{prompt}} = \text{[CLS]}\ \boldsymbol{x}\ .\ \underbrace{\text{It was}}_{\text{pattern}}\ \text{[MASK]}\ .\ \text{[SEP]}$$

Then, the PLM determines which *verbalizer* (e.g., 'great' and 'terrible') is the most likely substitute for the mask in the $\boldsymbol{x}_{\text{prompt}}$. This subsequently determines the score of targets ('positive' or 'negative'). In detail:

Training strategy Let $\mathcal{M}: \mathcal{Y} \rightarrow \mathcal{V}$ be a mapping from target labels to individual words in a PLM's vocabulary. We refer to this mapping as *verbalizers*. Then the input is converted to $\boldsymbol{x}_{\text{prompt}} = \mathcal{T}(\boldsymbol{x})$ by appending a *pattern* and a *mask token* to $\boldsymbol{x}$ so that it has the format of a masked language modeling input. Then, the classification task is converted to a MLM objective (Tam et al., 2021; Schick and Schütze, 2021a), and the PLM computes the probability of the label y as:

$$\begin{aligned} p(y|\boldsymbol{x}) &= p(\text{[MASK]} = \mathcal{M}(y)|\boldsymbol{x}_{\text{prompt}}) \\ &= \frac{\exp(\boldsymbol{W}_{\mathcal{M}(y)}^T \boldsymbol{h}_{\text{[MASK]}})}{\sum_{v' \in \mathcal{V}} \exp(\boldsymbol{W}_{v'}^T \boldsymbol{h}_{\text{[MASK]}})}, \end{aligned} \tag{6.2}$$

where $\boldsymbol{h}_{\text{[MASK]}}$ is the last hidden representation of the mask, and $\boldsymbol{W}_v$ shows the output embedding of the PLM for each verbalizer $v \in \mathcal{V}$. For many tasks, verbalizers have multiple tokens. Schick and Schütze (2021b) extended (6.2) to multiple mask tokens by adding the maximum number of mask

tokens M needed to express the outputs (verbalizers) for a task. In that case, Schick and Schütze (2021b) computes the probability of each class as the summation of the log probabilities of each token in the corresponding verbalizer, and then they add a hinge loss to ensure a margin between the correct verbalizer and the incorrect ones.

Inference strategy During inference, the model needs to select which verbalizer to use in the given context. Schick and Schütze (2021b) predicts the verbalizer tokens in an autoregressive fashion. They first trim the number of mask tokens from M to each candidate verbalizer's token length and compute the probability of each mask token. They then choose the predicted token with the highest probability and replace the corresponding mask token. Conditioning on this new token, the probabilities of the remaining mask positions are recomputed. They repeat this autoregressive decoding until they fill all mask positions. This inference strategy is very slow, as the number of forward passes increases with the number of classes and the number of verbalizer's tokens.

This formulation obtained impressive few-shot performance with PLMs. However, the success of this approach heavily relies on engineering handcrafted *patterns* and *verbalizers*. Coming up with suitable verbalizers and patterns can be difficult (Mishra et al., 2021). Additionally, the performance is sensitive to the wording of patterns (Zhao et al., 2021; Perez et al., 2021; Schick and Schütze, 2021a; Jiang et al., 2020) or to the chosen verbalizers (Webson and Pavlick, 2021).

In addition, handcrafted verbalizers cause problems for efficient training: a) they require updating the PLM embedding layer, causing large memory overhead; b) fine-tuning PLMs also requires a very small learning rate (usually 10^{-5}), which slows down tuning the parameters of the verbalizers; c) modeling verbalizers as one of the tokens of the PLM vocabulary (perhaps unintentionally) impacts the input representation during tuning; d) verbalizers have variable token lengths, complicating the implementation in a vectorized format, thereby making it challenging to efficiently fine-tune PLMs.

6.3 Method

We propose PERFECT, a *verbalizer and pattern free* few-shot learning method. We design PERFECT to be close to the pre-training phase, similar to the PET family of models (Schick and Schütze, 2021b; Gao et al., 2021), while replacing handcrafted patterns and verbalizers with new components that are designed to describe the task and learn the labels. As shown in Figure 6.3, we first convert each input $\boldsymbol{x}_{\text{input}}$ to its masked language modeling (MLM) input containing M mask tokens [MASK][1] with no added patterns, denoted as $\boldsymbol{x}_{\text{masked}} = \mathcal{T}(\boldsymbol{x}_{\text{input}})$.[2] PERFECT then trains a classifier per-token and optimizes the average multi-class hinge loss over each mask position.

Three main components play a role in the success of PERFECT: a) a pattern-free task description, where we use task-specific adapters to efficiently tell the model about the given task, replacing previously

[1] We discuss the general case with inserting multiple masks; for some datasets this improves performance (§6.4.3).

[2] We insert mask tokens after the input string in single-sentence benchmarks, and after the first sentence in the case of sentence-pair datasets and encode both sentences as a single input, which we found to perform the best (Appendix 6.9).

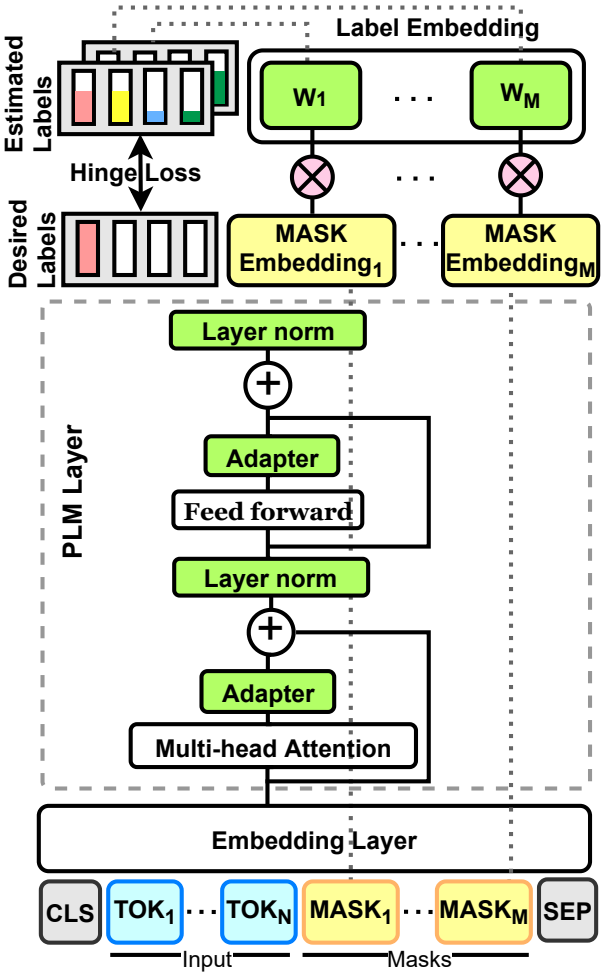

Figure 6.3: We remove handcrafted patterns and verbalizers. We replace patterns using task-specific adapters and design label embeddings for the classes. We only train the green blocks (the label embeddings, adapters, and layer norms).

manually engineered patterns (§6.3.1), b) multi-token label-embedding as an efficient mechanism to learn the label representations, removing manually designed verbalizers (§6.3.2). c) an efficient inference strategy building on top of the idea of prototypical networks (Snell et al., 2017) (§6.3.4), which replaces prior iterative autoregressive decoding methods (Schick and Schütze, 2021b).

As shown in Figure 6.3, we fix the underlying PLM model and only optimize the new parameters that we add (green boxes). This includes the task-specific adapters to adapt the representations for a given task and the multi-token label representations. We detail each of these components below.

6.3.1 Pattern-Free Task Description

We use task-specific adapter layers to provide the model with learned, implicit task descriptions. Adapters additionally bring multiple other benefits: a) fine-tuning all weights of PLMs with millions or

billions of parameters is sample-inefficient, and can be unstable in low-resource settings (Dodge et al., 2020); adapters allow sample-efficient fine-tuning, by keeping the underlying PLM fixed, b) adapters reduce the storage and memory footprints (§6.4.2), c) they also increase stability and performance (§6.4), making them an excellent choice for few-shot fine-tuning. To our knowledge, this is the first approach for using *task-specific adapters* to effectively and efficiently remove patterns in few-shot learning. Experimental results in §6.4 show its effectiveness compared to handcrafted patterns and soft prompts (Li and Liang, 2021; Lester et al., 2021).

6.3.2 Multi-Token Label Embeddings

We freeze the weights of the PLM's embedding layer and introduce a separate label embedding $\boldsymbol{L} \in \mathbb{R}^{K \times M \times H}$, which is a multi-token label representation where M is the number of tokens representing each label, K indicates the number of classes, H is the input hidden dimension. Using a fixed number of tokens M for each label, versus variable-token length verbalizers used in prior work (Schick and Schütze, 2021a,b) substantially simplifies the implementation and accelerates the training (§6.4.2).

6.3.3 Training PERFECT

As shown in Figure 6.3, we optimize label embeddings so that the PLM predicts the correct label, and optimize adapters to adapt the PLM for the given task. For label embeddings, PERFECT trains a classifier per token and optimizes the average multi-class hinge loss over all mask positions. Given $\boldsymbol{x}_{\text{masked}}$, let $\boldsymbol{h}_{[\text{MASK}]_i}$ be the embedding of its i-th mask token from the last layer of the PLM encoder. Additionally, let $f(.): \mathbb{R}^H \rightarrow \mathbb{R}^K$ be a per-token classifier that computes the predictions by multiplying the mask token embedding with its corresponding label embedding. Formally defined as:

$$\boldsymbol{t_i} = f(\boldsymbol{h}_{[\text{MASK}]_i}) = \boldsymbol{L}_i^T \boldsymbol{h}_{[\text{MASK}]_i},$$

where $\boldsymbol{L_i} \in \mathbb{R}^{K \times H}$ shows the label embedding for the i-th mask position. Then, for each mask position, we optimize a multi-class hinge loss between their scores $\boldsymbol{t_i}$ and labels. Formally defined as:

$$\mathcal{L}(\boldsymbol{x}, y, i) = \frac{\sum_{k=1, k \neq y}^{K} \max(0, m - \boldsymbol{t_{iy}} + \boldsymbol{t_{ik}})}{K},$$

where $\boldsymbol{t_{ik}}$ shows the k-th element of $\boldsymbol{t_i}$, representing the score corresponding to class k, and m is the margin, which we fix to the default value of $m = 1$. Then, the final loss is computed by averaging the loss over all mask tokens and training samples:

$$\mathcal{L} = \frac{1}{M|\mathcal{D}|} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{x}, y, i) \tag{6.3}$$

6.3.4 Inference with PERFECT

During evaluation, instead of relying on the prior iterative autoregressive decoding schemes (Schick and Schütze, 2021b), we classify a query point by finding the nearest class prototype to the mask token embeddings:

$$y = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \max_{i \in \{1,\ldots,M\}} \left(\exp^{-d(\boldsymbol{h}_i^q, \boldsymbol{c_{iy}})} \right), \tag{6.4}$$

where d is squared euclidean distance,[3] $\boldsymbol{h}_i^q$ indicates the embedding of the i-th mask position for the query sample q, and $\boldsymbol{c_{iy}} \in \mathbb{R}^D$ is the prototype representation of the i-th mask token with class label y, i.e., the mean embedding of i-th mask position in all training samples with label y:

$$\boldsymbol{c_{iy}} = \frac{1}{|\mathcal{D}_y|} \sum_{b \in \mathcal{D}_y} \boldsymbol{h}_i^b, \tag{6.5}$$

where $\boldsymbol{h}_i^b$ shows the embedding of i-th mask position for training sample b, and $\mathcal{D}_y$ is the training instances with class y. This strategy closely follows prototypical networks (Snell et al., 2017), but applied across multiple tokens. We choose this form of inference because prototypical networks are known to be sample efficient and robust (Snell et al., 2017), and because it substantially speeds up evaluation compared to prior methods (§6.4.2).

6.4 Experiments

We conduct extensive experiments on a variety of NLP datasets to evaluate the performance of PERFECT and compare it with state-of-the-art few-shot learning.

Datasets We consider 7 tasks and 12 datasets: 1) the sentiment analysis datasets SST-2 (Socher et al., 2013), SST-5 (Socher et al., 2013), MR (Pang and Lee, 2005), and CR (Hu and Liu, 2004), 2) the subjectivity classification dataset SUBJ (Pang and Lee, 2004), 3) the question classification dataset TREC (Voorhees and Tice, 2000), 4) the natural language inference datasets CB (De Marneffe et al., 2019) and RTE (Wang et al., 2019b), 5) the question answering dataset QNLI (Rajpurkar et al., 2016), 6) the word sense disambiguation dataset WiC (Pilehvar and Camacho-Collados, 2019), 7) the paraphrase detection datasets MRPC (Dolan and Brockett, 2005) and QQP.[4] See datasets statistics in Appendix 6.7.

For MR, CR, SST-5, SUBJ, and TREC, we test on the original test sets, while for other datasets, since test sets are not publicly available, we test on the original validation set. We sample 16 instances per label from the training set to form training and validation sets.

Baselines We compare with the state-of-the-art few-shot learning of PET and fine-tuning:

[3] We also tried with cosine similarity but found a slight improvement with squared Euclidean distance (Snell et al., 2017).
[4] https://quoradata.quora.com/

Method	SST-2	CR	MR	SST-5	Subj	TREC	Avg
Single-Sentence Benchmarks							
FINETUNE	81.4/70.0/4.0	80.1/72.9/4.1	77.7/66.8/4.6	39.2/34.3/2.5	**90.2**/84.1/**1.8**	87.6/75.8/3.7	76.0/67.3/3.4
PET-Average	89.7/81.0/2.4	88.4/68.8/3.0	85.9/79.0/2.1	45.9/40.3/**2.4**	88.1/79.6/2.4	85.0/70.6/4.5	80.5/69.9/2.8
PET-Best	89.1/81.0/2.6	88.8/85.8/1.9	**86.4/82.0/1.6**	**46.0**/41.2/2.4	88.7/**84.6/1.8**	85.8/70.6/4.4	80.8/74.2/2.4
Logan IV et al. (2021)	89.8/84.1/1.7	89.9/87.2/1.1	84.9/76.2/3.2	45.7/**41.6/2.3**	81.8/73.5/4.0	84.7/81.8/1.6	79.5/74.1/2.3
PERFECT-rand	90.7/**88.2/1.2**	90.0/85.5/1.4	86.3/81.4/**1.6**	42.7/35.1/2.9	89.1/82.8/2.1	**90.6**/81.6/3.2	**81.6/75.8/2.1**
Ablation							
PERFECT-init	**90.9**/87.6/1.5	89.7/87.4/1.2	85.4/75.8/3.3	42.8/35.9/3.5	87.6/81.6/2.8	90.4/**86.6/1.8**	81.1/**75.8**/2.4
prompt+mte	70.6/56.0/8.3	71.0/55.8/8.2	66.6/49.6/7.3	32.2/26.5/3.2	82.7/69.6/3.9	79.6/66.8/6.5	67.1/54.0/6.2
bitfit+mte	89.5/81.7/3.0	**90.1/87.8/1.0**	85.6/80.5/1.9	42.3/36.8/3.3	89.1/82.4/2.4	90.4/85.0/1.4	81.2/75.7/2.2
Method	**CB**	**RTE**	**QNLI**	**MRPC**	**QQP**	**WiC**	**Avg**
Sentence-Pair Benchmarks							
FINETUNE	72.9/67.9/**2.5**	56.8/50.2/**3.5**	62.7/51.4/7.0	**70.1/62.7/4.7**	65.0/59.8/3.6	52.4/46.1/3.7	63.3/56.4/4.2
PET-Average	86.9/73.2/5.1	60.1/49.5/4.7	66.5/55.7/6.2	62.1/38.2/6.8	63.4/44.7/7.9	51.0/46.1/2.6	65.0/51.2/5.6
PET-Best	90.0/78.6/3.9	62.3/51.3/4.5	70.5/57.9/6.4	63.4/49.3/6.5	70.7/55.2/5.8	51.6/47.2/2.3	68.1/56.6/4.9
Logan IV et al. (2021)	91.0/87.5/2.7	**64.4/58.5**/3.9	71.2/**66.5/2.6**	63.9/53.7/5.3	70.4/62.7/**3.4**	52.4/**48.4/1.8**	68.9/**62.9/3.3**
PERFECT-rand	**90.3/83.9**/3.5	60.4/53.1/4.7	**74.1**/60.3/4.6	67.8/54.7/5.7	**71.2**/64.2/3.5	**53.8**/47.0/3.0	**69.6**/60.5/4.2
Ablation							
PERFECT-init	87.9/75.0/4.9	60.7/52.7/4.5	72.8/56.7/6.8	65.9/56.6/6.0	71.1/**65.6**/3.5	51.7/46.6/2.8	68.4/58.9/4.8
prompt+mte	73.0/62.5/6.1	56.9/50.7/4.1	55.4/50.2/4.6	60.0/51.5/5.8	54.3/46.2/5.6	51.3/46.7/2.8	58.5/51.3/4.8
bitfit+mte	89.6/82.1/4.3	61.3/53.8/5.2	70.6/51.9/5.9	68.5/57.4/5.1	69.4/63.0/3.9	52.9/47.8/2.7	68.7/59.3/4.5

Table 6.1: Performance of all methods on single-sentence and sentence-pair benchmarks. We report average/worst-case accuracy/standard deviation. PERFECT obtains the state-of-the-art results. Bold fonts indicate the best results.

PET (Schick and Schütze, 2021a,b) is the state-of-the-art few-shot learning method that employs carefully crafted verbalizers and patterns. We report the best (PET-best) and average (PET-average) results among all patterns and verbalizers.[5]

FINETUNE The standard fine-tuning (Devlin et al., 2019), with adding a classifier on top of the [CLS] token and fine-tuning all parameters.

Our method We study the performance of PERFECT and perform an extensive ablation study to show the effectiveness of our design choices:

PERFECT-rand We randomly initialize the label embedding $\boldsymbol{L}$ from a normal distribution $\mathcal{N}(0,\sigma)$ with $\sigma = 10^{-4}$ (chosen based on validation performance, see Appendix 6.10) *without relying on any handcrafted patterns and verbalizers*. As an ablation, we study the following two variants:

[5]For a controlled study, we use the MLM variant shown in (6.2), which has been shown to perform the best (Tam et al., 2021).

PERFECT-init We initialize the label embedding with the token embeddings of manually designed verbalizers in the PLM's vocabulary to study the impact of engineered verbalizers.

prompt+mte To compare the impact of adapters versus soft prompt-tuning for few-shot learning, we append trainable continuous prompt embeddings to the input (Lester et al., 2021). Then we only tune the soft prompt and multi-token label embeddings (mte).

bitfit+mte Following Cai et al. (2020) and Ravfogel et al. (2021), we tune biases as an alternative to adapters. We additionally tune multi-token label embeddings.

Logan IV et al. (2021) Following Logan IV et al. (2021), we remove patterns and tune the biases in the PET.

Experimental details We use the RoBERTa large model (Liu et al., 2019b) (355M parameters) as the underlying PLM for all methods. We use the HuggingFace PyTorch implementation (Wolf et al., 2020). For the baselines, we used the carefully manually designed patterns and verbalizers in Gao et al. (2021), Min et al. (2021), and Schick and Schütze (2021b) (usually 5 different options per datasets; see Appendix 6.8).

We evaluate all methods using 5 different random samples to create the training/validation sets and 4 different random seeds for training. Therefore, for PET-average, we report the results on 20 x 5 (number of patterns and verbalizers) = 100 runs, while for PET-best and our method, we report the results over 20 runs. The variance in few-shot learning methods is usually high (Perez et al., 2021; Zhao et al., 2021; Lu et al., 2021). Therefore, we report average, worst-case performance, and standard deviation across all runs, where the last two values can be important for risk-sensitive applications (Asri et al., 2016).

6.4.1 Experimental Results

Table 6.1 shows the performance of all methods. PERFECT obtains state-of-the-art results, improving the performance compared to PET-average by +1.1 and +4.6 points for single-sentence and sentence-pair datasets respectively. It even outperforms PET-best, where we report the best performance of PET across multiple manually engineered patterns and verbalizers. Moreover, PERFECT generally improves the minimum performance and reduces standard deviation substantially. Finally, PERFECT is also significantly more efficient: reducing the training and inference time, memory usage, and storage costs (see §6.4.2).

PET-best improves the results over PET-average showing that PET is unstable to the choice of patterns and verbalizers; this difference is more severe for sentence-pair benchmarks. This might be because the position of the mask highly impacts the results, and the patterns used for sentence-pair datasets in Schick and Schütze (2021b) exploits this variation by putting the mask in multiple locations (see Appendix 6.8).

Removing patterns and tuning biases in Logan IV et al. (2021) is not expressive enough and performs substantially worse than PERFECT on average.

Metric	PET	PERFECT	Δ%
Trained params (M)	355.41	**3.28**	**-99.08%**
Peak memory (GB)	20.93	**16.34**	**-21.93%**
Training time (min)	23.42	**0.65**	**-97.22%**
+ PET in batch	0.94	**0.65**	**-30.85%**
Inference time (min)	9.57	**0.31**	**-96.76%**

Table 6.2: Percentage of trained parameters, average peak memory, training, and inference time. $\Delta\%$ is the relative difference with respect to PET. Lower is better.

As an ablation, even if we initialize the label embedding with handcrafted verbalizers in PERFECT-init, it consistently obtains lower performance, demonstrating that PERFECT is able to obtain state-of-the-art performance with learning from *pure random initialization*. We argue that initializing randomly close to zero (with low variance $\sigma = 10^{-4}$), as done in our case, slightly improves performance, which perhaps is not satisfied when initializing from the manually engineered verbalizers (see Appendix 6.10).

As a second ablation, when learning patterns with optimizing soft prompts in prompt+mte, we observe high sensitivity to learning rate, as also confirmed in Li and Liang (2021) and Mahabadi et al. (2021a). We experimented with multiple learning rates but performance consistently lags behind PERFECT-rand. This can be explained by the low flexibility of such methods as all the information regarding specifying patterns needs to be contained in the prefixes. As a result, the method only allows limited interaction with the rest of the model parameters, and obtaining good performance requires very large models (Lester et al., 2021). In addition, increasing the sequence length leads to memory overhead (Mahabadi et al., 2021a), and the number of prompt tokens is capped by the number of tokens that can fit in the maximum input length, which can be a limitation for tasks requiring large contexts.

As a third ablation, tuning biases with optimizing soft prompts in bitfit+mte obtains lower performance compared to PERFECT, showing that adapters are a better alternative compared to tuning biases to learn task descriptions for few-shot learning.

We include more ablation results on design choices of PERFECT in Appendix 6.11.

6.4.2 Efficiency Evaluation

In this section, we compare the efficiency of PERFECT with the state-of-the-art few-shot learning method, PET. To this end, we train all methods for ten epochs on the 500-sampled QNLI dataset. We select the largest batch size for each method that fits a fixed budget of the GPU memory (40 GB).

Due to the auto-regressive inference strategy of PET (Schick and Schütze, 2021b), all prior work implemented it with a batch size of 1 (Perez et al., 2021; Schick and Schütze, 2021b; Tam et al., 2021). Additionally, since PET deals with verbalizers of variable lengths, it is hard to implement their training phase in batch mode. We specifically choose QNLI to have verbalizers of the same length and enable batching for comparison purposes (referred to as *PET in batch*). However, verbalizers are still not of

Dataset	PET-Average	Pattern-Free
SST-2	89.7/81.0/2.4	**90.5/87.8/1.2**
CR	88.4/68.8/3.0	**89.8/87.0/1.4**
MR	85.9/79.0/2.1	**86.4/83.0/1.8**
SST-5	**45.9/40.3/2.4**	44.8/40.0/2.4
SUBJ	**88.1/79.6/2.4**	85.3/74.7/3.8
TREC	85.0/70.6/4.5	**87.9/84.6/1.8**
CB	86.9/73.2/5.1	**93.0/89.3/1.9**
RTE	60.1/49.5/4.7	**63.7/56.3/4.1**
QNLI	66.5/55.7/6.2	**71.3/65.8/2.5**
MRPC	62.1/38.2/6.8	**66.0/54.4/5.6**
QQP	63.4/44.7/7.9	**71.8/64.3/3.7**
WiC	51.0/46.1/2.6	**53.7/50.3/2.0**
Avg	72.8/60.6/4.2	**75.4/69.8/2.7**

Table 6.3: Average performance of *PET* with five different patterns vs. *Pattern-Free* that replaces handcrafted patterns with task-specific adapters. We report the average/worst-case performance/and the standard deviation.

fixed-length for most other tasks, and this speed-up does not apply generally to PET.

In Table 6.2, for each method we report the percentage of trained parameters, memory usage, training time, and inference time. PERFECT reduces the number of trained parameters, and therefore the storage requirement, by 99.08%. It additionally reduces the memory requirement by 21.93% compared to PET. PERFECT speeds up training substantially, by 97.22% relative to the original PET's implementation, and 30.85% to our implementation of PET. This is because adapter-based tuning saves on memory and allows training with larger batch sizes. In addition, PERFECT is significantly faster during inference time (96.76% less inference time relative to PET).

Note that although prompt+mte and bitfit+mte can also reduce the storage costs, by having 0.02M and 0.32 M trainable parameters respectively, they are not expressive enough to learn task descriptions, and their performance substantially lags behind PERFECT (see Table 6.1).

Overall, given the size of PLMs with millions and billions of parameters (Liu et al., 2019b; Raffel et al., 2020), efficient few-shot learning methods are of paramount importance for practical applications. PERFECT not only outperforms the state-of-the-art in terms of accuracy and generally improves the stability (Table 6.1), but also is significantly more efficient in runtime, storage, and memory.

6.4.3 Analysis

Can task-specific adapters replace manually engineered patterns? PERFECT is a pattern-free approach and employs adapters to provide the PLMs with task descriptions implicitly. In this section, we study the contribution of replacing manual patterns with adapters in isolation without considering

Dataset	PERFECT	-Adapters
SST-2	**90.7/88.2/1.2**	88.2/81.9/2.3
CR	**90.0/85.5/1.4**	89.2/83.1/1.7
MR	**86.3/81.4/1.6**	82.5/78.2/2.5
SST-5	**42.7/35.1/2.9**	40.6/33.6/3.3
SUBJ	89.1/82.8/2.1	**89.7/85.0/1.9**
TREC	**90.6/81.6/3.2**	89.8/74.2/4.3
CB	**90.3/83.9**/3.5	89.6/**83.9/2.8**
RTE	60.4/53.1/**4.7**	**61.7/53.8**/5.1
QNLI	**74.1/60.3/4.6**	73.2/56.3/5.8
MRPC	67.8/**54.7/5.7**	**68.0**/54.2/6.1
QQP	**71.2/64.2/3.5**	71.0/62.0/3.7
WiC	**53.8/47.0/3.0**	52.5/46.9/3.0
Avg	**75.6/68.1/3.1**	74.7/66.1/3.5

Table 6.4: Performance of PERFECT w/o adapters, *-Adapters*. We report the average performance/worst-case performance/and the standard deviation.

our other contributions in representing labels, training, and inference. In PET (Schick and Schütze, 2021a,b), we replace the handcrafted patterns with task-specific adapters (*Pattern-Free*) while keeping the verbalizers and the training and inference intact[6] and train it with a similar setup as in §6.4. Table 6.3 shows the results. While PET is very sensitive to the choice of prompts, adapters provide an efficient alternative to learn patterns robustly by improving the performance (average and worst-case) and reducing the standard deviation. This finding demonstrates that task-specific adapters can effectively replace manually engineered prompts. Additionally, they also save on the training budget by at least $1/$number of patterns (normally 1/5) by not requiring running the method for different choices of patterns, and by freezing most parameters, this saves on memory and offers additional speed-up.

Impact of Removing Adapters To study the impact of adapters in learning patterns, we remove adapters, while keeping the label embedding. Handcrafted patterns are not included and we tune all parameters of the model. Table 6.4 shows the results. Adding adapters for learning patterns contributes to the performance by improving the average performance, and making the model robust by improving the minimum performance and reducing the standard deviation. This is because training PLMs with millions of parameters is sample-inefficient and unstable on resource-limited datasets (Dodge et al., 2020; Zhang et al., 2021b; Mosbach et al., 2021). However, by using adapters, we substantially reduce the number of trainable parameters, allowing the model to be better tuned in a few-shot setting.

Impact of the number of masks In Table 6.1, to compare our design with PET in isolation, we fixed the number of mask tokens as the maximum number inserted by PET. In table 6.5, we study the impact

[6]Since we don't have patterns, in the case of multiple sets of verbalizers, we use the first set of verbalizers as a random choice.

Datasets	1	2	5	10
CR	90.1	**90.2**	89.0	87.8
MR	**86.9**	86.1	85.4	85.6
MRPC	67.4	68.2	70.1	**72.3**
QNLI	73.7	**73.9**	73.0	65.1
RTE	**60.0**	57.3	56.2	56.0
TREC	90.0	**90.9**	88.9	88.8
Avg	**78.0**	77.8	77.1	75.9

Table 6.5: Test performance for the varying number of mask tokens. Bold fonts indicate the best results in each row.

of varying the number of inserted mask tokens for a random selection of six tasks. For most tasks, having two mask tokens performs the best, while for MR and RTE, having one, and for MRPC, inserting ten masks improves the results substantially. The number of required masks might be correlated with the difficulty of the task. PERFECT is designed to be general, enabling having multiple mask tokens.

6.5 Related Work

Adapter Layers Mahabadi et al. (2021b) and Üstün et al. (2020) proposed to generate adapters' weights using hypernetworks (Ha et al., 2017), where Mahabadi et al. (2021b) proposed to share a small hypernetwork to generate conditional adapter weights efficiently for each transformer layer and task. Mahabadi et al. (2021a) proposed compacter layers by building on top of ideas of parameterized hyper-complex layers (Zhang et al., 2021a) and low-rank methods (Li et al., 2018; Aghajanyan et al., 2021), as an efficient fine-tuning method for PLMs. We are the first to employ adapters to replace handcrafted patterns for few-shot learning.

Few-shot Learning with PLMs Le Scao and Rush (2021) showed that prompting provides substantial improvements compared to fine-tuning, especially in low-resource settings. Subsequently, researchers continuously tried to address the challenges of manually engineered patterns and verbalizers: a) Learning the patterns in a continuous space (Li and Liang, 2021; Qin and Eisner, 2021; Lester et al., 2021), while freezing PLM for efficiency, has the problem that, in most cases, such an approach only works with very large scale PLMs (Lester et al., 2021), and lags behind full fine-tuning in a general setting, while being inefficient and not as effective compared to adapters (Mahabadi et al., 2021a). b) Optimizing patterns in a discrete space (Shin et al., 2020; Jiang et al., 2020; Gao et al., 2021) has the problem that such methods are computationally costly. c) Automatically finding verbalizers in a discrete way Schick et al. (2020); Schick and Schütze (2021a) is computationally expensive and does not perform as well as manually designed ones. d) Removing manually designed patterns (Logan IV et al., 2021) substantially lags behind the expert-designed ones. Our proposed method, PERFECT, does not rely on any handcrafted patterns and verbalizers.

6.6 Conclusion

We proposed PERFECT, a simple and efficient method for few-shot learning with pre-trained language models without relying on handcrafted patterns and verbalizers. PERFECT employs task-specific adapters to learn task descriptions implicitly, replacing previous handcrafted patterns, and a continuous multi-token label embedding to represent the output classes. Through extensive experiments over 12 NLP benchmarks, we demonstrate that PERFECT, despite being far simpler and more efficient than recent few-shot learning methods, produces state-of-the-art results. Overall, the simplicity and effectiveness of PERFECT make it a promising approach for few-shot learning with PLMs.

Appendix

6.7 Experimental Details

Datasets Table 6.6 shows the stastistics of the datasets used. We download SST-2, MR, CR, SST-5, and SUBJ from Gao et al. (2021), while the rest of the datasets are downloaded from the HuggingFace Datasets library (Lhoest et al., 2021). RTE, CB, WiC datasets are from SuperGLUE benchmark (Wang et al., 2019b), while QQP, MRPC and QNLI are from GLUE benchmark (Wang et al., 2019c) with Creative Commons license (CC BY 4.0). RTE (Wang et al., 2019b) is a combination of data from RTE1 (Dagan et al., 2005), RTE2 (Bar-Haim et al., 2006), RTE3 (Giampiccolo et al., 2007), and RTE5 (Bentivogli et al., 2009). For WiC (Pilehvar and Camacho-Collados, 2019) sentences are selected from VerbNet (Schuler, 2005), WordNet (Miller, 1995), and Wiktionary.

Computing infrastructure We run all the experiments on one NVIDIA A100 with 40G of memory.

Training hyper-parameters We set the maximum sequence length based on the recommended values in the HuggingFace repository (Wolf et al., 2020) and prior work (Min et al., 2021; Schick and Schütze, 2021b), i.e., we set it to 256 for SUBJ, CR, CB, RTE, and WiC, and 128 for other datasets. For all methods, we use a batch size of 32. For FINETUNE and PET, we use the default learning rate of 10^{-5}, while for our method, as required by adapter-based methods (Mahabadi et al., 2021a), we set the learning rate to a higher value of 10^{-4}.[7] Through all experiments, we fix the adapter bottleneck size to 64. Following Pfeiffer et al. (2021), we experimented with keeping one of the adapters in each layer for better training efficiency and found keeping the adapter after the feed-forward module in each layer to perform the best. For tuning label embedding, we use the learning rate of $\{10^{-1},10^{-2},10^{-3},10^{-4},10^{-5}\}$ and choose the one obtaining the highest validation performance. For PERFECT-prompt, we tune the continuous prompt for learning rate of $\{10^{-1},10^{-2},10^{-3}\}$.[8]Following Lester et al. (2021), for PERFECT-prompt, we set the number of prompt tokens to 20, and initialize them with a random subset of the top 5000 token's embedding of the PLM. We train all methods for 6000 steps. Based on our results, this is sufficient to allow the models to converge. We save a checkpoint every 100 steps for all methods and report the results for the hyper-parameters performing the best on the validation set for each task.

[7]We have also tried to tune the baselines with the learning rate of 10^{-4} but it performed worst.

[8]We also tried tuning prompts with learning rates of $\{10^{-4},10^{-5}\}$ but it performed worst, as also observed in prior work (Mahabadi et al., 2021a; Min et al., 2021).

Dataset	Task	#Train	#Test	K
	Single-Sentence Benchmarks			
MR	Sentiment analysis	8662	2000	2
CR	Sentiment analysis	1774	2000	2
SST-2	Sentiment analysis	6920	872	2
SST-5	Sentiment analysis	8544	2210	5
SUBJ	Subjectivity classification	8000	2000	2
TREC	Question classification	5452	500	6
	Sentence-Pair Benchmarks			
CB	Natural language inference	250	56	3
RTE	Natural language inference	2490	277	2
WiC	Word sense disambiguation	5428	638	2
MRPC	Paraphrase detection	3668	408	2
QNLI	Question answering	104743	5463	2
QQP	Paraphrase detection	363846	40430	2

Table 6.6: Statistics of datasets used in this work. We sample $N \times |\mathcal{Y}|$ instances (with multiple seeds) from the original training set to form the few-shot training and validation sets. The test column shows the size of the test set.

6.8 Choice of Patterns and Verbalizers

For SST-2, MR, CR, SST-5, and TREC, we used 4 different patterns and verbalizers from Gao et al. (2021). For CB, WiC, RTE datasets, we used the designed patterns and verbalizers in Schick and Schütze (2021b). For QQP, MRPC, and QNLI, we wrote the patterns and verbalizers inspired by the ones in Schick and Schütze (2021b). The used patterns and verbalizers are as follows:

- For sentiment analysis tasks (**MR**, **CR**, **SST-2**, **SST-5**), given a sentence s:

  s A <MASK> one.

  s It was <MASK>.

  s All in all <MASK>.

  s A <MASK> piece.

  with "great" as a verbalizer for positive, "terrible" for negative. In case of SST-5 with five labels, we expand it to "great", "good", "okay", "bad", and "terrible".

- For **SUBJ**, given a sentence s:

s This is <MASK>.

s It's all <MASK>.

s It's <MASK>.

s Is it <MASK>?

with "subjective" and "objective" as verbalizers.

- For **TREC**, given a question q, the task is to classify the type of it:

q <MASK>:

q Q:<MASK>:

q why<MASK>?

q Answer: <MASK>.

with "Description", "Entity", "Expression", "Human", "Location", "Number" as verbalizers for question types of "Description", "Entity", "Abbreviation", "Human", "Location", and "Numeric".

- For entailment task (**RTE**) given a premise p and hypothesis h:

"h" ? | <MASK>, "p"

h? | <MASK>, p

"h" ? | <MASK>. p

with "Yes" as a verbalizer for entailment, "No" for contradiction.

p question: h True or False? answer: <MASK>

with "true" as a verbalizer for entailment, "false" for contradiction.

- For entailment task (**CB**) given a premise p and a hypothesis h:

"h" ? | <MASK>, "p"

h? | <MASK>, p

"h" ? | <MASK>. p

with "Yes" as a verbalizer for entailment, "No" for contradiction, "Maybe" for neutral.

p question: h true, false or neither? answer: <MASK>

with "true" as a verbalizer for entailment, "false" for contradiction, "neither" for neutral.

- For **QNLI**, given a sentence s and question q:

s. Question: q? Answer: <MASK>.

with "Yes" or "true" as verbalizers for entailment and "No" or "false" for not entailment.

s. Based on the previous sentence, q? <MASK>.

with "Yes" or "true" as verbalizers for entailment and "No" or "false" for not entailment.

Based on the following sentence, q?<MASK>.s

with "Yes" and "No" as verbalizers for entailment and not entailment respectively.

- For **QQP**, given two questions q_1 and q_2:

Do q_1 and q_2 have the same meaning?<MASK>.

with "Yes" or "true" as verbalizers for duplicate and "No" or "false" for not duplicate.

q_1. Based on the previous question, q_2? <MASK>.

with "Yes" or "true" as verbalizers for duplicate and "No" or "false" for not duplicate.

Based on the following question, q_1?<MASK>.q_2

with "Yes" and "No" as verbalizers for duplicate and not duplicate respectively.

- For **MRPC**, given two sentences s_1 and s_2:

Do s_1 and s_2 have the same meaning?<MASK>.

with "Yes" or "true" as verbalizers for equivalent and "No" or "false" for not equivalent.

> s_1. Based on the previous sentence, s_2? <MASK>.

with "Yes" or "true" as verbalizers for equivalent and "No" or "false" for not equivalent.

> Based on the following sentence, s_1?<MASK>.s_2

with "Yes" and "No" as verbalizers for equivalent and not equivalent respectively.

- For **WiC**, given two sentences s_1 and s_2 and a word w, the task is to classify whether w is used in the same sense.

> "s_1" / "s_2". Similar sense of "w"? <MASK>.

> s_1 s_2 Does w have the same meaning in both sentences? <MASK>

With "No" and "Yes" as verbalizers for False, and True.

> w . Sense (1) (a) "s_1" (<MASK>) "s_2"

With "2" and "b" as verbalizers for False, and True.

6.9 Impact of the Position of Masks in Sentence-pair Datasets

We evaluate the impact of the position of mask tokens in sentence-pair benchmarks. Given two sentences s_1 and s_2, we consider the following four locations for inserting mask tokens, where in the case of encoding as two sentences, input parts to the encoder are separated with |:

1. s_1 s_2 <MASK>
2. s_1 <MASK> s_2
3. s_1 | <MASK> s_2
4. s_1 | s_2<MASK>

Table 6.7 shows how the position of masks impact the results. As demonstrated, pattern 2, inserting mask tokens between the two sentences and encoding both as a single sentence obtains the highest validation performance. We use this choice in all the experiments when removing handcrafted patterns.

6.10 Impact of Initialization

We initialize the label embedding matrix with random initialization from a normal distribution $\mathcal{N}(0,\sigma)$. In table 6.8, we show the development results for different values of σ. We choose the σ obtaining

Datasets	1	2	3	4
CB	89.8	**91.6**	88.9	86.5
RTE	**69.1**	**69.1**	64.5	65.3
QNLI	72.0	**83.3**	77.7	73.1
MRPC	71.6	69.5	66.4	**72.0**
QQP	79.2	**82.8**	72.5	70.2
WiC	**60.3**	59.5	60.2	59.5
Avg	73.7	**76.0**	71.7	71.1

Table 6.7: Validation performance for sentence-pair benchmarks for different locations of mask tokens. Bold fonts indicate the best results in each row.

Datasets	10^{-2}	10^{-3}	10^{-4}	10^{-5}
CB	90.0/82.5	92.2/85.0	91.6/87.5	91.6/87.5
MRPC	69.8/56.2	70.8/56.2	69.5/56.2	70.8/56.2
QNLI	83.3/71.9	82.7/71.9	83.3/71.9	83.1/68.8
QQP	82.8/78.1	82.7/75.0	82.8/75.0	83.0/75.0
RTE	69.8/62.5	69.2/59.4	69.1/62.5	68.3/62.5
WiC	62.2/50.0	59.7/46.9	59.5/53.1	58.9/50.0
Avg	76.3/66.9	76.2/65.7	76.0/67.7	76.0/66.7
Total Avg	71.6	71.0	**71.8**	71.3

Table 6.8: Validation performance for different values of σ. We show mean performance/worst-case performance across 20 runs. The last row shows the average of mean performance/worst-case performance.

the highest performance on average over average and worst case performance, i.e., $\sigma = 10^{-4}$.

6.11 Ablation Results

To study the impact of different design choices in PERFECT, we considered the following experiments:

- **-Hinge Loss:** In this variant, we replace the hinge loss with multi-class cross entropy loss.
- **+Label Emb:** We use the trained label embeddings during the inference, substituting the computed prototypes in (6.5).
- **-Prototypical:** Instead of using prototypical networks, during inference, we use the same objective as training, i.e., (6.4).

Results are shown in Table 6.9. Experimental results demonstrate that PERFECT obtains the best results on average. Using multi-class cross-entropy instead of hinge loss, obtains substantially lower minimum performance (67.4 versus 68.1), demonstrating that training with hinge loss makes the

Dataset	PERFECT	-Hinge Loss	+Label Emb	-Prototypical
SST-2	**90.7/88.2**/1.2	90.0/85.9/1.7	90.6/87.6/**1.1**	90.4/85.2/1.6
CR	90.0/85.5/1.4	**90.1/88.6/0.9**	89.7/86.6/1.4	89.9/86.8/1.4
MR	**86.3**/81.4/1.6	85.2/78.6/2.4	85.8/**82.4/1.4**	85.7/78.0/2.0
SST-5	42.7/35.1/2.9	**43.3**/36.8/3.1	41.8/**37.1**/2.5	41.2/35.9/**2.4**
SUBJ	89.1/82.8/2.1	89.4/83.1/2.2	**90.0/86.0/1.8**	89.7/**86.0/1.8**
TREC	**90.6/81.6/3.2**	89.9/76.8/4.2	89.7/71.6/6.1	89.6/76.2/4.9
CB	**90.3/83.9/3.5**	89.2/80.4/4.8	89.6/82.1/3.6	89.3/80.4/3.9
RTE	60.4/53.1/4.7	**60.7/54.5/4.0**	58.6/50.9/4.0	58.5/50.9/4.5
QNLI	74.1/60.3/4.6	72.9/64.4/3.9	**74.9**/66.7/3.6	74.7/**67.5/3.5**
MRPC	67.8/54.7/5.7	67.0/49.8/5.5	**68.1/56.9/4.8**	**68.1/56.9/4.8**
QQP	**71.2/64.2/3.5**	69.9/63.0/4.1	70.3/62.2/4.0	70.2/62.2/4.0
WiC	**53.8**/47.0/3.0	53.7/46.7/3.3	53.6/**50.2/2.4**	53.6/50.0/2.6
Avg	**75.6/**68.1/**3.1**	75.1/67.4/3.3	75.2/**68.4/3.1**	75.1/68.0/**3.1**

Table 6.9: Ablation results on the impact of different design choices in PERFECT. We report the average performance/worst-case performance/and the standard deviation.

model more stable. Using the trained label embeddings (+Label Emb) obtains very close results to PERFECT (slightly worse on average and slightly better on the minimum performance). Using the similar objective as training with replacing prototypical networks (-Prototypical), obtains lower performance on average (75.1 versus 75.6). These results confirm the design choices for PERFECT.

7 Conclusions

This chapter summarizes the contributions of this thesis and suggests possible directions for future research. This thesis has reported progress in five important aspects of transfer learning in NLP:

- Training models robust to biases in datasets
- Reducing overfitting, especially in a low-resource setting
- Learning from multiple resources and generalizing to unseen domains
- Efficient fine-tuning methods for pretrained language models
- Few-shot learning with pretrained language models

In Chapter 2, we propose two novel techniques, product-of-experts, and debiased focal loss, to reduce biases learned by neural models, which are applicable whenever one can specify the biases in the form of one or more bias-only models. The bias-only models are designed to leverage biases and shortcuts in the datasets. Our debiasing strategies then work by adjusting the cross-entropy loss based on the performance of these bias-only models, to focus learning on the hard examples and down-weight the importance of the biased examples. Additionally, we extend our methods to combat multiple bias patterns simultaneously. Our proposed debiasing techniques are model agnostic, simple, and highly effective. Extensive experiments show that our methods substantially improve the model robustness to domain-shift, including 9.8 points gain on FEVER symmetric test set, 7.4 on the HANS dataset, and 4.8 points on SNLI hard set. Furthermore, we show that our debiasing techniques result in better generalization to other NLI datasets. Future work may include developing debiasing strategies that do not require prior knowledge of bias patterns and can automatically identify them.

In Chapter 3, we propose VIBERT, an effective model to reduce overfitting when fine-tuning large-scale pretrained language models on low-resource datasets. By leveraging a VIB objective, VIBERT finds the simplest sentence embedding, predictive of the target labels, while removing task-irrelevant and redundant information. Our approach is model agnostic, simple to implement, and highly effective.

Extensive experiments and analyses show that our method substantially improves transfer performance in low-resource scenarios. We demonstrate our obtained sentence embeddings are robust to biases and our model results in a substantially better generalization to out-of-domain NLI datasets. Future work includes exploring incorporating VIB on multiple layers of pretrained language models and using it to jointly learn relevant features and relevant layers.

In Chapter 4, we propose a parameter-efficient method for multi-task fine-tuning. Our approach is to train shared hypernetworks to generate task-specific adapters conditioned on the task, layer id, and adapter position embeddings. The shared hypernetworks capture the knowledge across tasks and enable positive transfer to low-resource and related tasks, while task-specific layers allow the model to adapt to each individual task. Extensive experiments show that our method obtains strong improvement over multi-task learning on the GLUE benchmark, and substantially improves the in-domain task generalization.

In Chapter 5, we have proposed COMPACTER, a light-weight fine-tuning method for large-scale language models. COMPACTER generates weights by summing Kronecker products between shared "slow" weights and "fast" rank-one matrices, specific to each COMPACTER layer. Leveraging this formulation, COMPACTER reduces the number of parameters in adapters substantially from $\mathcal{O}(kd)$ to $\mathcal{O}(k+d)$. Through extensive experiments, we demonstrate that despite learning 2127.66$\times$ fewer parameters than standard fine-tuning, COMPACTER obtains comparable or better performance in a full-data setting and outperforms fine-tuning in data-limited scenarios.

In Chapter 6, we proposed PERFECT, a simple and efficient method for few-shot learning with pretrained language models without relying on handcrafted patterns and verbalizers. PERFECT employs task-specific adapters to learn task descriptions implicitly, replacing previous handcrafted patterns, and a continuous multi-token label embedding to represent the output classes. Through extensive experiments over 12 NLP benchmarks, we demonstrate that PERFECT, despite being far simpler and more efficient than recent few-shot learning methods, produces state-of-the-art results. Overall, the simplicity and effectiveness of PERFECT make it a promising approach for few-shot learning with pretrained language models. PERFECT handles a wide variety of classification tasks; extending PERFECT to generation tasks, such as machine translation and summarization, is an interesting future direction.

In summary, this thesis has contributed a variety of methods for more effective transfer learning. Transfer learning is a crucial topic in NLP, radically improving the state-of-the-art results in a variety of benchmarks (Raffel et al., 2020; Chowdhery et al., 2022; Du et al., 2022; Brown et al., 2020; Hoffmann et al., 2022).

There is still a wide range of issues for future work that are not touched in this thesis, such as coming up with more complex benchmarks to evaluate the performance of language models (Wang et al., 2019b), collecting more clean unlabeled datasets for better pretraining (Raffel et al., 2020), improving the efficiency of language models (Dettmers et al., 2022; Rajbhandari et al., 2019; Smith et al., 2022), coming up with better pretraining objectives (Dai and Le, 2015; Ramachandran et al., 2017; Radford et al., 2018; Devlin et al., 2019; Yang et al., 2019; Liu et al., 2019a; Wang et al., 2019a; Song et al.,

2019; Dong et al., 2019; Joshi et al., 2020) allowing to capture general-purpose knowledge from the unlabeled data, and many others.

This thesis contributes to progress towards the goal of achieving efficient neural models of general-purpose language understanding for any NLP task.

Bibliography

Aghajanyan, A., Zettlemoyer, L., and Gupta, S. (2021). Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *ACL*.

Alemi, A., Fischer, I., Dillon, J., and Murphy, K. (2017). Deep variational information bottleneck. In *ICLR*.

Arase, Y. and Tsujii, J. (2019). Transfer fine-tuning: A bert case study. In *EMNLP*.

Arivazhagan, N., Bapna, A., Firat, O., Lepikhin, D., Johnson, M., Krikun, M., Chen, M. X., Cao, Y., Foster, G., Cherry, C., et al. (2019). Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint arXiv:1907.05019*.

Arora, S., Cohen, N., and Hazan, E. (2018). On the optimization of deep networks: Implicit acceleration by overparameterization. In *ICML*.

Asri, H., Mousannif, H., Al Moatassime, H., and Noel, T. (2016). Using machine learning algorithms for breast cancer risk prediction and diagnosis. *Procedia Computer Science*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Baldridge, J., He, L., and Zhang, Y. (2019). Paws: Paraphrase adversaries from word scrambling. In *NAACL*.

Bansal, T., Jha, R., and McCallum, A. (2020). Learning to Few-Shot Learn Across Diverse Natural Language Classification Tasks. In *COLING*.

Bapna, A. and Firat, O. (2019). Simple, scalable adaptation for neural machine translation. In *EMNLP*.

Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., and Giampiccolo, D. (2006). The second pascal recognising textual entailment challenge. *Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Belinkov, Y., Poliak, A., Shieber, S., Van Durme, B., and Rush, A. (2019a). Don't take the premise for granted: Mitigating artifacts in natural language inference. In *ACL*.

Belinkov, Y., Poliak, A., Shieber, S. M., Van Durme, B., and Rush, A. M. (2019b). On adversarial removal of hypothesis-only bias in natural language inference. In *SEM*.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *JMLR*.

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., and Magnini, B. (2009). The fifth pascal recognizing textual entailment challenge. In *TAC*.

Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*.

Bowman, S., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space. In *CoNLL*.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *EMNLP*.

Brock, A., De, S., Smith, S. L., and Simonyan, K. (2021). High-performance large-scale image recognition without normalization. *ICML*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *NeurIPS*.

Cadene, R., Dancette, C., Ben-younes, H., Cord, M., and Parikh, D. (2019). Rubi: Reducing unimodal biases in visual question answering. In *NeurIPS*.

Cai, H., Gan, C., Zhu, L., and Han, S. (2020). Tinytl: Reduce memory, not parameters for efficient on-device learning. *NeurIPS*.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. (2017). Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *SemEval*.

Chelba, C. and Acero, A. (2004). Adaptation of maximum entropy capitalizer: Little data can help a lot. In *EMNLP*.

Chen, Q., Zhu, X., Ling, Z.-H., Wei, S., Jiang, H., and Inkpen, D. (2017). Enhanced lstm for natural language inference. In *ACL*.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. (2020). The lottery ticket hypothesis for pre-trained bert networks. *NeurIPS*.

Cherry, C., Durrett, G., Foster, G., Haffari, R., Khadivi, S., Peng, N., Ren, X., and Swayamdipta, S., editors (2019). *DeepLo*.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury,

J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. (2022). Palm: Scaling language modeling with pathways.

Chung, H. W., Févry, T., Tsai, H., Johnson, M., and Ruder, S. (2021). Rethinking embedding coupling in pre-trained language models. In *ICLR*.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019a). BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*.

Clark, C., Yatskar, M., and Zettlemoyer, L. (2019b). Don't take the easy way out: Ensemble based methods for avoiding known dataset biases. In *EMNLP*.

Clark, K., Luong, M.-T., Khandelwal, U., Manning, C. D., and Le, Q. (2019c). Bam! born-again multi-task networks for natural language understanding. In *ACL*.

Conneau, A. and Kiela, D. (2018). Senteval: An evaluation toolkit for universal sentence representations. In *LREC*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. In *EMLP*.

Dagan, I., Glickman, O., and Magnini, B. (2005). The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. *NeurIPS*.

Daumé III, H. (2007). Frustratingly easy domain adaptation. In *ACL*.

De Marneffe, M.-C., Simons, M., and Tonhauser, J. (2019). The commitmentbank: Investigating projection in naturally occurring discourse. In *proceedings of Sinn und Bedeutung*.

De Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., and Courville, A. C. (2017). Modulating early visual processing by language. In *NeurIPS*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*.

Desai, S., Zhan, H., and Aly, A. (2019). Evaluating lottery tickets under distributional shifts. In *DeepLo*.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). Llm.int8(): 8-bit matrix multiplication for transformers at scale.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Dodge, J., Gururangan, S., Card, D., Schwartz, R., and Smith, N. A. (2019). Show your work: Improved reporting of experimental results. In *EMNLP-IJCNLP*.

Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. (2020). Fine-tuning pre-trained language models: Weight initializations, data orders, and early stopping. *arXiv:2002.06305*.

Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *IWP*.

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M., and Hon, H.-W. (2019). Unified language model pre-training for natural language understanding and generation. *arXiv preprint arXiv:1905.03197*.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., Zoph, B., Fedus, L., Bosma, M. P., Zhou, Z., Wang, T., Wang, E., Webster, K., Pellat, M., Robinson, K., Meier-Hellstern, K., Duke, T., Dixon, L., Zhang, K., Le, Q., Wu, Y., Chen, Z., and Cui, C. (2022). GLaM: Efficient scaling of language models with mixture-of-experts. In *ICML*.

Elazar, Y. and Goldberg, Y. (2018). Adversarial removal of demographic attributes from text data. In *EMNLP*.

Fadaee, M., Bisazza, A., and Monz, C. (2017). Data augmentation for low-resource neural machine translation. In *ACL*.

Gao, T., Fisch, A., and Chen, D. (2021). Making pre-trained language models better few-shot learners. *ACL*.

Gaudet, C. J. and Maida, A. S. (2018). Deep quaternion networks. In *IJCNN*.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. (2007). The third PASCAL recognizing textual entailment challenge. In *ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*.

Gong, Y., Luo, H., and Zhang, J. (2017). Natural language inference over interaction space. In *ICLR*.

Grand, G. and Belinkov, Y. (2019). Adversarial regularization for visual question answering: Strengths, shortcomings, and side effects. In *Proceedings of the Second Workshop on Shortcomings in Vision and Language*.

Gururangan, S., Dang, T., Card, D., and Smith, N. A. (2019). Variational pretraining for semi-supervised text classification. In *ACL*.

Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S., and Smith, N. A. (2018). Annotation artifacts in natural language inference data. In *NAACL*.

Ha, D., Dai, A., and Le, Q. V. (2017). Hypernetworks. In *ICLR*.

Hambardzumyan, K., Khachatrian, H., and May, J. (2021). Warp: Word-level adversarial reprogramming. *ACL*.

Harremoës, P. and Tishby, N. (2007). The information bottleneck revisited or how to choose a good distortion measure. In *ISIT*.

He, H., Zha, S., and Wang, H. (2019). Unlearn dataset bias in natural language inference by fitting the residual. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 132–142, Hong Kong, China. Association for Computational Linguistics.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. (2017). Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., Driessche, G. v. d., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. (2022). Training compute-optimal large language models.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *ICML*.

Howard, J. and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. In *ACL*.

Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *SIGKDD*.

Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. (2018). GPipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*.

Huh, M., Agrawal, P., and Efros, A. A. (2016). What makes ImageNet good for transfer learning? *arXiv preprint arXiv:1608.08614*.

Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S., and Hofmann, K. (2019). Generalization in reinforcement learning with selective noise injection and information bottleneck. In *NeurIPS*.

Jia, R. and Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems. In *EMNLP*.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*.

Jiang, Z., Xu, F. F., Araki, J., and Neubig, G. (2020). How can we know what language models know? *TACL*.

Jin, T., Liu, Z., Yan, S., Eichenberger, A., and Morency, L.-P. (2020). Language to network: Conditional parameter adaptation with natural language descriptions. In *ACL*.

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., and Levy, O. (2020). Spanbert: Improving pre-training by representing and predicting spans. *TACL*.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Kaushik, D. and Lipton, Z. C. (2018). How much reading does reading comprehension require? a critical investigation of popular benchmarks. In *EMNLP*.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. (2019). CTRL: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Khashabi, D., Chaturvedi, S., Roth, M., Upadhyay, S., and Roth, D. (2018). Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *NAACL*.

Khot, T., Sabharwal, A., and Clark, P. (2018). Scitail: A textual entailment dataset from science question answering. In *AAAI*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. In *ICLR*.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *NeurIPS*.

Lai, A., Bisk, Y., and Hockenmaier, J. (2017). Natural language inference from multiple premises. In *IJCNLP*.

Le, Q., Sarlós, T., and Smola, A. (2013). Fastfood-approximating kernel expansions in loglinear time. In *ICML*.

Le, T., Bertolini, M., Noé, F., and Clevert, D.-A. (2021). Parameterized hypercomplex graph neural networks for graph classification. *ICANN*.

Le Scao, T. and Rush, A. M. (2021). How many data points is a prompt worth? In *NAACL*.

Lee, C., Cho, K., and Kang, W. (2019). Mixout: Effective regularization to finetune large-scale pretrained language models. In *ICLR*.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. *EMNLP*.

Levesque, H., Davis, E., and Morgenstern, L. (2012). The winograd schema challenge. In *KR*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

Lhoest, Q., Villanova del Moral, A., Jernite, Y., Thakur, A., von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., Davison, J., Šaško, M., Chhablani, G., Malik, B., Brandeis, S., Le Scao, T., Sanh, V., Xu, C., Patry, N., McMillan-Major, A., Schmid, P., Gugger, S., Delangue, C., Matussière, T., Debut, L., Bekman, S., Cistac, P., Goehringer, T., Mustar, V., Lagunas, F., Rush, A., and Wolf, T. (2021). Datasets: A community library for natural language processing. In *EMNLP*.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018). Measuring the intrinsic dimension of objective landscapes. In *ICLR*.

Li, X. and Roth, D. (2002). Learning question classifiers. In *COLING*.

Li, X. L. and Eisner, J. (2019). Specializing word embeddings (for parsing) by information bottleneck. In *EMNLP*.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *ACL*.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*.

Lin, Z., Madotto, A., and Fung, P. (2020). Exploring versatile generative language model via parameter-efficient transfer learning. In *EMNLP Findings*.

Liu, X., He, P., Chen, W., and Gao, J. (2019a). Multi-task deep neural networks for natural language understanding. In *ACL*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Logan IV, R. L., Balažević, I., Wallace, E., Petroni, F., Singh, S., and Riedel, S. (2021). Cutting down on prompts and parameters: Simple few-shot learning with language models. *arXiv preprint arXiv:2106.13353*.

Lu, Y., Bartolo, M., Moore, A., Riedel, S., and Stenetorp, P. (2021). Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning word vectors for sentiment analysis. In *ACL*.

Mahabadi, K. R., Belinkov, Y., and Henderson, J. (2020). End-to-end bias mitigation by modelling biases in corpora. In *ACL*.

Mahabadi, R. K., Henderson, J., and Ruder, S. (2021a). Compacter: Efficient low-rank hypercomplex adapter layers. In *NeurIPS*.

Mahabadi, R. K., Ruder, S., Dehghani, M., and Henderson, J. (2021b). Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *ACL*.

Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. (2018). Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., and Zamparelli, R. (2014). A sick cure for the evaluation of compositional distributional semantic models. In *LREC*.

McCann, B., Bradbury, J., Xiong, C., and Socher, R. (2017). Learned in translation: Contextualized word vectors. *NeurIPS*.

McCann, B., Keskar, N. S., Xiong, C., and Socher, R. (2018). The natural language decathlon: Multitask learning as question answering. *arXiv:1806.08730*.

McCoy, R. T., Min, J., and Linzen, T. (2019a). Berts of a feather do not generalize together: Large variability in generalization across models with similar test set performance. *arXiv preprint arXiv:1911.02969*.

McCoy, T., Pavlick, E., and Linzen, T. (2019b). Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *NeurIPS*.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*.

Min, S., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. (2021). Noisy channel language model prompting for few-shot text classification. *arXiv preprint arXiv:2108.04106*.

Mishra, S., Khashabi, D., Baral, C., and Hajishirzi, H. (2021). Cross-task generalization via natural language crowdsourcing instructions.

Mosbach, M., Andriushchenko, M., and Klakow, D. (2021). On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines. *ICLR*.

Mou, L., Men, R., Li, G., Xu, Y., Zhang, L., Yan, R., and Jin, Z. (2016). Natural language inference by tree-based convolution and heuristic matching. In *ACL*.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*.

Oswald, J. V., Henning, C., Sacramento, J., and Grewe, B. F. (2020). Continual learning with hypernetworks. In *ICLR*.

Pang, B. and Lee, L. (2004). A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *ACL*.

Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*.

Parcollet, T., Ravanelli, M., Morchid, M., Linarès, G., Trabelsi, C., De Mori, R., and Bengio, Y. (2018a). Quaternion recurrent neural networks. In *ICLR*.

Parcollet, T., Zhang, Y., Morchid, M., Trabelsi, C., Linarès, G., de Mori, R., and Bengio, Y. (2018b). Quaternion convolutional neural networks for end-to-end automatic speech recognition. In *Interspeech*.

Pavlick, E. and Callison-Burch, C. (2016). Most "babies" are "little" and most "problems" are "huge": Compositional entailment in adjective-nouns. In *ACL*.

Pavlick, E., Wolfe, T., Rastogi, P., Callison-Burch, C., Dredze, M., and Van Durme, B. (2015). Framenet+: Fast paraphrastic tripling of framenet. In *ACL*.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *EMNLP*.

Perez, E., Kiela, D., and Cho, K. (2021). True few-shot learning with language models. *NeurIPS*.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *AAAI*.

Peters, M. E., Neumann, M., Zettlemoyer, L., and Yih, W.-t. (2018). Dissecting contextual word embeddings: Architecture and representation. In *EMNLP*.

Peters, M. E., Ruder, S., and Smith, N. A. (2019). To tune or not to tune? adapting pretrained representations to diverse tasks. In *RepL4NLP*.

Pfeiffer, J., Kamath, A., RückĹe, A., Kyunghyun, C., and Gurevych, I. (2021). AdapterFusion: Non-destructive task composition for transfer learning. In *EACL*.

Pfeiffer, J., Rücklé, A., Poth, C., Kamath, A., Vulić, I., Ruder, S., Cho, K., and Gurevych, I. (2020). AdapterHub: A framework for adapting transformers. In *EMNLP: System Demonstrations*.

Phang, J., Févry, T., and Bowman, S. R. (2018). Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv:1811.01088*.

Pilault, J., hattami, A. E., and Pal, C. (2021). Conditionally adaptive multi-task learning: Improving transfer learning in NLP using fewer parameters & less data. In *ICLR*.

Pilehvar, M. T. and Camacho-Collados, J. (2019). Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *NAACL*.

Platanios, E. A., Sachan, M., Neubig, G., and Mitchell, T. (2018). Contextual parameter generation for universal neural machine translation. In *EMNLP*.

Poliak, A., Naradowsky, J., Haldar, A., Rudinger, R., and Van Durme, B. (2018). Hypothesis only baselines in natural language inference. In *SEMEVAL*.

Ponti, E. M., Vulić, I., Cotterell, R., Parovic, M., Reichart, R., and Korhonen, A. (2020). Parameter space factorization for zero-shot learning across tasks and languages. *arXiv preprint arXiv:2001.11453*.

Prasanna, S., Rogers, A., and Rumshisky, A. (2020). When BERT Plays the Lottery, All Tickets Are Winning. In *EMNLP*.

Qin, G. and Eisner, J. (2021). Learning how to ask: Querying lms with mixtures of soft prompts. In *NAACL*.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Rahman, A. and Ng, V. (2012). Resolving complex cases of definite pronouns: the winograd schema challenge. In *EMNPL*.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2019). Zero: Memory optimization towards training A trillion parameter models.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP*.

Ramachandran, P., Liu, P. J., and Le, Q. (2017). Unsupervised pretraining for sequence to sequence learning. In *EMNLP*.

Ramakrishnan, S., Agrawal, A., and Lee, S. (2018). Overcoming language priors in visual question answering with adversarial regularization. In *NeurIPS*.

Ratner, A., Hancock, B., Dunnmon, J., Goldman, R., and Ré, C. (2018). Snorkel metal: Weak supervision for multi-task learning. In *DEEM workshop*.

Ravfogel, S., Ben-Zaken, E., and Goldberg, Y. (2021). Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked languagemodels. *arXiv preprint arXiv:2106.10199*.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *CVPR*.

Reisinger, D., Rudinger, R., Ferraro, F., Harman, C., Rawlins, K., and Van Durme, B. (2015). Semantic proto-roles. In *TACL*.

Roemmele, M., Bejan, C. A., and Gordon, A. S. (2011). Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI Symposium Series*.

Rücklé, A., Geigle, G., Glockner, M., Beck, T., Pfeiffer, J., Reimers, N., and Gurevych, I. (2021). AdapterDrop: On the Efficiency of Adapters in Transformers. *EMNLP*.

Ruder, S. (2017). An Overview of Multi-Task Learning in Deep Neural Networks. In *arXiv preprint arXiv:1706.05098*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International journal of computer vision*.

Schick, T., Schmid, H., and Schütze, H. (2020). Automatically identifying words that can serve as labels for few-shot text classification. In *COLING*.

Schick, T. and Schütze, H. (2021a). Exploiting cloze-questions for few-shot text classification and natural language inference. In *EACL*.

Schick, T. and Schütze, H. (2021b). It's not just size that matters: Small language models are also few-shot learners. In *NAACL*.

Schuler, K. K. (2005). Verbnet: A broad-coverage, comprehensive verb lexicon. *PhD Thesis*.

Schuster, T., J Shah, D., Jie Serene Yeo, Y., Filizzola, D., Santus, E., and Barzilay, R. (2019). Towards debiasing fact verification models. In *EMNLP*.

Shamir, O., Sabato, S., and Tishby, N. (2010). Learning and generalization with the information bottleneck. In *TCS*.

Sharma, R., Allen, J., Bakhshandeh, O., and Mostafazadeh, N. (2018). Tackling the story ending biases in the story cloze test. In *ACL*.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., Sepassi, R., and Hechtman, B. (2018). Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Shin, T., Razeghi, Y., Logan IV, R. L., Wallace, E., and Singh, S. (2020). Eliciting knowledge from language models using automatically generated prompts. In *EMNLP*.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., Zhang, E., Child, R., Aminabadi, R. Y., Bernauer, J., Song, X., Shoeybi, M., He, Y., Houston, M., Tiwary, S., and Catanzaro, B. (2022). Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *NeurIPS*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2019). Mass: Masked sequence to sequence pre-training for language generation. In *ICML*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. In *JMLR*.

Stickland, A. C. and Murray, I. (2019). Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *ICML*.

Tam, D., Menon, R. R., Bansal, M., Srivastava, S., and Raffel, C. (2021). Improving and simplifying pattern exploiting training. *arXiv preprint arXiv:2103.11955*.

Tay, Y., Zhang, A., Luu, A. T., Rao, J., Zhang, S., Wang, S., Fu, J., and Hui, S. C. (2019). Lightweight and efficient neural natural language processing with quaternion networks. In *ACL*.

Tay, Y., Zhao, Z., Bahri, D., Metzler, D., and Juan, D.-C. (2021). Hypergrid transformers: Towards a single model for multiple tasks. In *ICLR*.

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism quarterly*.

Tishby, N., Pereira, F. C., and Bialek, W. (1999). The information bottleneck method. In *Allerton*.

Üstün, A., Bisazza, A., Bouma, G., and van Noord, G. (2020). Udapter: Language adaptation for truly universal dependency parsing. In *EMNLP*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*.

Voita, E., Sennrich, R., and Titov, I. (2019). The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *EMNLP-IJCNLP*.

Voorhees, E. M. and Tice, D. M. (2000). Building a question answering test collection. In *SIGIR*.

Vu, T., Wang, T., Munkhdalai, T., Sordoni, A., Trischler, A., Mattarella-Micke, A., Maji, S., and Iyyer, M. (2020). Exploring and predicting transferability across NLP tasks. In *EMNLP*.

Wang, A., Hula, J., Xia, P., Pappagari, R., McCoy, R. T., Patel, R., Kim, N., Tenney, I., Huang, Y., Yu, K., et al. (2019a). Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. In *ACL*.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019b). SuperGLUE: a stickier benchmark for general-purpose language understanding systems. In *NeurIPS*.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2019c). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Wang, S., Li, B., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Wang, Z., Dai, Z., Póczos, B., and Carbonell, J. (2019d). Characterizing and avoiding negative transfer. In *CVPR*.

Wang, Z., Hamza, W., and Florian, R. (2017). Bilateral multi-perspective matching for natural language sentences. In *IJCAI*.

Warstadt, A., Singh, A., and Bowman, S. R. (2019). Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*.

Webson, A. and Pavlick, E. (2021). Do prompt-based models really understand the meaning of their prompts? *arXiv preprint arXiv:2109.01247*.

Wen, Y., Tran, D., and Ba, J. (2020). BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning. In *ICLR*.

West, P., Holtzman, A., Buys, J., and Choi, Y. (2019). Bottlesum: Unsupervised and self-supervised sentence summarization using the information bottleneck principle. In *EMNLP*.

White, A. S., Rastogi, P., Duh, K., and Van Durme, B. (2017). Inference is everything: Recasting semantic resources into a unified evaluation framework. In *IJCNLP*.

Williams, A., Nangia, N., and Bowman, S. (2018). A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*.

Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *EMNLP: System Demonstrations*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

Yin, W., Rajani, N. F., Radev, D., Socher, R., and Xiong, C. (2020). Universal natural language processing with limited annotations: Try few-shot textual entailment as a start. In *EMNLP*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *NeurIPS*.

Zhang, A., Tay, Y., Zhang, S., Chan, A., Luu, A. T., Hui, S., and Fu, J. (2021a). Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with 1/n parameters. In *ICLR*.

Zhang, S., Liu, X., Liu, J., Gao, J., Duh, K., and Van Durme, B. (2018). Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.

Zhang, S., Rudinger, R., Duh, K., and Van Durme, B. (2017). Ordinal common-sense inference. In *TACL*.

Zhang, T., Wu, F., Katiyar, A., Weinberger, K. Q., and Artzi, Y. (2021b). Revisiting Few-sample BERT Fine-tuning. In *ICLR*.

Zhang, X., Zhao, J., and Lecun, Y. (2015). Character-level convolutional networks for text classification. In *NeurIPS*.

Zhao, T. Z., Wallace, E., Feng, S., Klein, D., and Singh, S. (2021). Calibrate before use: Improving few-shot performance of language models. *ICML*.

Zhu, X., Xu, Y., Xu, H., and Chen, C. (2018). Quaternion convolutional neural networks. In *ECCV*.

Rabeeh Karimi Mahabadi

Personal information

Address Office 306, Idiap Research Institute, Rue Marconi 19, 1920 Martigny, Switzerland

Phone +41 - 783163448

Email rkarimi@idiap.ch, rabeeh.karimimahabadi@epfl.ch

Webpage https://www.idiap.ch/~rkarimi/

Education

PhD in NLP, Idiap Research Institute/EPFL Lausanne, *Supervisors: Prof. James Henderson, Prof. Volkan Cevher*, GPA: 5.75/6, Mar 2017 - till present.
With focus on improving generalization of language models, few-shot learning, multi-task learning.

Master of Computer Science, ETH Zurich, *GPA: 5.49/6*, Sep 2013 - Sep 2016.

With strong focus on machine learning and computer vision

BSc of Electical Engineering, *Amirkabir University of Technology, Iran*, GPA: 17.77/20, Sep 2008 - Sep 2012.

Ranked 1st according to GPA among all B.Sc. students

Pre-university Certificate, Mathematics and physics Discipline, *Aboureihan Educational Complex, Iran*, GPA: 18.73/20, Sep 2007 - Sep 2008.

Work experience

DeepMind, mentored by Charles Blundell, Andrea Banino, Tim Scholtes, Research scientist intern in language team, March 2022-July 2022.

Meta Research, London/US, remotely in Switzerland, *Research Scientist contingent worker*, January 2022-current, Few-shot Generalization of Language models, In collaboration with: Majid Yazdani.

Meta Research, London/US, remotely in Switzerland, *Research Intern*, Aug 2021-Dec 2021, Few-shot Generalization of Language models, In collaboration with: Majid Yazdani, Luke Zettlemoyer, Ves Stoyanov, Lambert Mathias, Marzieh saeidi, I developed an automatic method for few-shot generalization of pretrained language models.

Google Research, Mountain View, remotely in Switzerland, *Research Intern*, Hinton's group, Oct 2020-Jan 2021, I developed an efficient method for multi-task fine-tuning of large-scale language models.

Google AI, Zurich, *Software Engineering Intern, Machine Intelligence Group*, May 2016-Sep 2016, Language used: C++, Implemented a distributed method to read unstructured data in tables showing up in Google question-answering system.

Publications

PERFECT: Prompt-free and Efficient Language Model Fine-Tuning, *R. Karimi Mahabadi, L. Zettlemoyer, J. Henderson, L.Mathias, M. Saeidi, V. Stoyanov, M. Yazdani*, ACL, 2022.

Compacter: Efficient Low-Rank HyperComplex Adapter Layers, *R. Karimi Mahabadi, J. Henderson, S. Ruder*, NeurIPS, 2021, Acceptance rate: **26%**.

Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks, *R. Karimi Mahabadi, S. Ruder, M. Dehghani, J. Henderson*, ACL, 2021, **Oral**. Acceptance rate: **21.3%**.

ParsiNLU: A Suite of Language Understanding Challenges for Persian, *Daniel Khashabi, Arman Cohan, Siamak Shakeri, Pedram Hosseini, Pouya Pezeshkpour, Malihe Alikhani, Moin Aminnaseri, Marzieh Bitaab, Faeze Brahman, Sarik Ghazarian, Mozhdeh Gheini, Arman Kabiri, Rabeeh Karimi Mahabadi, Omid Memarrast, Ahmadreza Mosallanezhad, Erfan Noury, Shahab Raji, Mohammad Sadegh Rasooli, Sepideh Sadeghi, Erfan Sadeqi Azer, Niloofar Safi Samghabadi, Mahsa Shafaei, Saber Sheybani, Ali Tazarv, Yadollah Yaghoobzadeh*, TACL, also accepted in EMNLP, 2021. Acceptance rate: **22.4%**.

Variational Information Bottleneck for Effective Low-Resource Fine-Tuning, *R. Karimi Mahabadi, Y. Belinkov, J. Henderson*, ICLR, 2021. Acceptance rate: **28.7%**.

End-to-End Bias Mitigation by Modelling Biases in Corpora, *R. Karimi Mahabadi, Y. Belinkov, J. Henderson*, ACL, 2020. Acceptance rate: **22.7%**.

Learning Entailment-Based Sentence Embeddings from Natural Language Inference, *R.Karimi Mahabadi, F. Mai, J. Henderson*, `https://openreview.net/forum?id=BkxackSKvH`, 2019.

A Learning-Based Framework for Quantized Compressed Sensing, *R. Karimi Mahabadi, J. Lin, V. Cevher*, IEEE Signal Processing Letters, 2018.

Real-time DCT Learning-based Reconstruction of Neural Signals, *R. Karimi Mahabadi, C. Aprile, V. Cevher*, EUSIPCO, 2018.

A Non-Euclidean Gradient Descent Framework for Non-Convex Matrix Factorization, *Y. Hsieh, Y. Kao, R. Karimi Mahabadi, A. Kyrillidis, and V. Cevher*, IEEE Transactions on Signal Processing, 2018.

Learning-Based Compressive MRI, *B. Gözcü, R. Karimi Mahabadi, Y. Li, E. Ilıcak, T. Cukur, J. Scarlett, and V.Cevher*, IEEE Transactions on Medical Imaging, 2017.

Segment Based 3D Object Shape Priors, *R. Karimi Mahabadi, C. Hane, M. Pollefeys*, CVPR, 2015. Acceptance rate: **25.0%**.

Scalable sparse covariance estimation via self-concordance, *A. Kyrillidis, R. Karimi Mahabadi, Q. Tran-Dinh, V. Cevher*, AAAI 2014.

Advanced MATLAB for Electrical Engineers: Neural Networks, Image processing, Genetic Algorithms, Fuzzy logic, and Digital Communication, *A. Alamdari, R. Karimi Mahabadi, A. Doosti, Z. Rajabi*, Negarandeye Danesh publisher, ISBN:978-600-6190-11-2.

Simulink for Engineers, *A. Alamdari, R. Karimi Mahabadi*, Negarandeye Danesh publisher, ISBN:978-600-6190-04-4.

Published MATLAB books are being used as a reference and being taught to students in MATLAB courses of the universities in Iran.

Awards and Honors

Meta compute credits for the FLORES 101 Large-Scale Multilingual Machine Translation at WMT $1500 USD, *2021*.

Google Cloud Credit Grant for my project on multi-task learning for amount of $5000, *2021*.

Fairness in AI award in Swiss Machine Learning Day, *2019*, EPFL Lausanne.

Awarded EDIC fellowship for the academic year 2016-2017, EPFL Lausanne.

Awarded ETH Scholarship for the academic year 2015 due to the strong academic performance.

Accepted to Google NLP PhD summit with Full Travel Grant, *Zurich, Switzerland*, 2019.

130 **Awarded travel grant by Google to take part in Grace Hopper**, *2018*.

Awarded a travel grant to attend WiML (Women in Machine Learning), *2017*.

Ranked 1st according to GPA among all B.Sc. students, *Electrical Engineering department (Signal and Systems Program), Amirkabir University of Technology, Iran*, 2012.

Awarded Fellowship of Exceptional Talent for the M.Sc. Program, *Electrical Engineering Department, Amirkabir University of Technology, Iran*, 2012.

Ranked within the top 0.2% in the National University Entrance Examination for the B.Sc. degree among more than 400,000 participants, *2008*.

Academic experience

Natural Language Understanding Group, Idiap Research Institute, *Oct 2018-present, Supervisors: Prof. James Henderson*, I am focused on improving generalization of large-scale language models specially on low-resource scenarios, bias-reduction, efficient fine-tuning of large-scale language models, and multi-task learning.

Information and Inference Systems Group, EPFL, *Mar 2017 - Sep 2018*, Supervisor: Prof. Cevher, Frameworks: TensorFlow, Keras, I designed sampling patterns for MRI and EEG signal reconstruction with deep learning and improved the speed of the reconstructions.

Learning and Adaptive Systems Group, ETH Zurich, *Supervisor: Prof. Andreas Krause*, Oct 2015 - Apr 2016, Languages used: Python, I developed a semi-supervised video segmentation and an active learning method in a united framework. The proposed algorithm works for general multi-label settings, is based on submodular optimization techniques and efficiently computes approximate probabilistic marginals over a video.

Research Assistant in Computer vision and Geometry Group, ETH Zurich, *Supervisor: Prof. Pollefeys*, Feb 2014 - Nov 2014, Languages used: C++, I developed a software for joint 3D reconstruction and segmentation of images. The proposed model improves the quality of segmentation and reconstructions, fixes the disconnected parts and holes seen in previous reconstructions, and automatically segments objects based on their geometries.

Summer Intern in Information and Inference Systems Group, EPFL, *Supervisors: Prof. Volkan Cevher, Anastasios T.Kyrillidis, Quoc Tran-Dinh*, Jun 2013 - Sep 2013, Languages used: C++, MATLAB, I developed and implemented a new optimization framework for sparse covariance estimation and phase retrieval problems using machine learning techniques. The method achieved state-of-the-art results on the benchmarks.

Research Assistant in Multimedia signal processing Lab, *Amirkabir University, Supervisors: Prof. Hamid Sheikhzadeh, Yalda Mohsenzadeh*, Languages used: C++, Jan 2013 - Sep 2013 , I implemented a sparse Bayesian machine learning algorithm for embedded feature selection for relevance vector machines regression.

Research Assistant in Machine learning Laboratory, *Amirkabir University, Supervisors: Prof. Saeed Shiry*, Sep 2011 - Mar 2012, Languages used: C++, MATLAB, I developed a new kernel-based classification method for multi-class settings.

Research Assistant in Automation and Intelligent Monitoring Systems Research Laboratory, *Amirkabir University, Supervisors: Prof. Amirhossein rezaei*, Sep 2010 - Mar 2011, Languages used: MATLAB, C++, AVR, Implementing a program on a robot that we have assembled which enabled it to recognize the auditory commands and act upon them by moving in the ordered direction..

Teaching Experience

Teaching Assistant: Artificial Intelligence master program, *Idiap Research Institute*, Summer 2019-2020.

Teaching Assistant: Programming and Numerical Analysis with C++, *Lecturer: Prof. Ali pourmohammad*, Sep 2012 - Mar 2013.

Technical skillset

Big data skills	Kubernetes, Docker, Spark, AWS, and GCP
deep learning frameworks	Good knowledge of Pytorch
C++, Python, SQL	Good knowledge of the language
Java, Bash, HTML,CSS	Basic knowledge of the language
Version control systems	Git, SVN

Languages

English: Fluent (TOEFL IBT: score 100/120), **Persian**: Mother tongue, **German:** B1-B2, **French:** A2