THE UNIVERSITY *of* EDINBURGH

University of Edinburgh

School of Informatics

Doctor of Philosophy

Parameter-efficient Transfer Learning for Pre-trained Transformers

by

Asa Cooper Stickland

April 2022

Abstract

In this thesis I will tackle a problem in machine learning and natural language processing (NLP) that I will refer to as 'parameter efficient transfer learning'. This involves taking 'general-purpose', large-scale models trained on huge amounts of data, and specializing them to a particular task, without changing the underlying model that much. A recent paradigm in machine learning has been to do large scale 'pre-training' of a model on unsupervised data before specializing to a particular task. Typically this means 'full fine-tuning' of pre-trained models by updating every parameter of the pre-trained model on the new task.

In this thesis we consider an alternative approach to full fine-tuning where we only update a subset of (or small number of additional) pre-trained model parameters, hence the term 'parameter-efficient' transfer learning, which can save on computation and storage space, unlock new capabilities, and in some situations outperform fine-tuning every parameter.

In the first section we consider parameter-efficient transfer learning on English classification tasks. Our first contribution is an approach to fine-tuning pre-trained models on multiple tasks simultaneously. Typical approaches underperform task-specific models due to a lack of capacity and interference between tasks. Our contribution is an approach to 'multi-task' learning where we introduce small task-specific modules for each task, which enable us to achieve the same performance as task-specific fine-tuning with only a fraction of the parameters.

This initial exploration was done on relatively small models compared to the current state-of-the-art, and did not cover a popular approach of freezing pre-trained model parameters and only training the small modules. In the second half of this section we address these limitations, contributing a survey of parameter-efficient approaches, showing which parameter-efficient architectures work the best as model scale increases, and detailing trade-offs between performance, memory-efficiency and other factors.

In the second section we consider applying parameter-efficient transfer learning approach to machine translation (MT), which involves modeling sequences rather than class labels and is multilingual rather than English-only. This means approaches designed for English classification can underperform. We explore adapting systems that have only been trained on an unsupervised objective (involving multilingual text but not machine translation) to the MT task. We were the first to apply parameter-efficient techniques to this problem. We explore which parts of the transformer sequence-to-sequence architecture are important to adapt, and what percentage of the original model we need to update to match fine-tuning every parameter.

In further chapters we contribute a new approach where we train independent 'adapters' (a particular parameter-efficient architecture) for source language, target language and 'domain' (i.e. legal text), allowing us to compose them in ways not seen during training. Finally, we contribute an extensive series of experiments on what matters for the performance of parameter-efficient methods on machine translation.

Lay summary

This PhD thesis focuses on improving the efficiency of large neural networks trained to perform Natural Language Processing (NLP) tasks, particularly large 'transformer' language models. To understand the significance of this research, it's helpful to first understand what language models are. Roughly speaking, a language model refers to a machine learning model designed to predict natural language. This might involve predicting the final word in a sentence given the previous words (essentially a fancy version of the auto-correct common on mobile phones), or predicting a 'masked' word in a sentence given the surrounding words.

An example of a large language model is BERT (which stands for Bidirectional Encoder Representations from Transformers). It's designed to understand the context of a word in a sentence, making it highly effective for tasks that require a good understanding of the properties of a sentence, such as classifying it into positive or negative sentiment. BERT and its counterparts, like GPT (Generative Pre-trained Transformer), work by analyzing huge amounts of text and learning to predict what words or phrases might come next in a sentence. ChatGPT, a variant of the GPT model, is an example of these advanced NLP systems. It uses a similar approach to BERT but is 'fine-tuned' for generating human-like text responses.

The research in this thesis is directly relevant to models like ChatGPT, as it addresses the fundamental challenge of improving the efficiency of adapting these large-scale models for new tasks. These models are made up of millions, sometimes billions, of parameters. Parameters in neural networks are like individual bits of knowledge or rules that the model uses to make decisions, they are the 'knobs' or settings we tune when learning from analyzing text. The more parameters a model has, the more it can learn but also the more 'memory' (essentially space on specific high-performance processors on computers) it consumes.

Many recent advances in NLP have come from scaling up the number of parameters in language models, however as they grow larger, they require significantly more computational resources and memory. This is where the thesis comes in. It explores ways to make BERT and similar models more efficient in terms of parameter usage when 'fine-tuning'. Since these models are initially trained on text from the web, they often need to be specialized for various tasks, which involves tweaking all of the parameters. By developing methods that can specialize these models without tweaking as many parameters, the models can be more easily fine-tuned, and maintain or even improve their performance without needing a proportional increase in memory for the fine-tuning process.

Some of these techniques are additionally 'modular', meaning they are separate from the original neural network. This means we can train separate 'German' and '(English) sentiment analysis' modules, then later combine them to perform German sentiment analysis. Much of this thesis is concerned with using parameter-efficient tuning for multilingual applications like machine translation, including using the above idea of modularity to add additional 'domains' like legal or medical text to machine translation systems.

In summary, this thesis contributes valuable insights into making large NLP models more memory-efficient. This advancement is crucial for expanding the capabilities and potential applications of AI in language understanding, directly impacting technologies like ChatGPT. Because the fine-tuning process is less resource intensive with the techniques explored and this thesis, we can more easily take a general purpose model and fine-tune it for many purposes. You could imagine personalizing a language model to every user of an application, or creating language-specific or domain-specific models that would work well in, say, German, or, say, law.

Acknowledgements

I'm grateful to the Centre for Doctoral Training in Data Science, the Engineering and Physical Sciences Research Council, the University of Edinburgh for providing the funding for my PHD. Iain Murray has consistently provided high-quality feedback and advice throughout the PhD, and was indispensable. I credit Iain with making this journey a (mostly) pleasurable experience. I also have to thank my coauthors: Xian Li, Marjan Ghazvininejad, Vassilina Nikoulina, Alexandre Berard and Ahmet Üstün, as well as everyone who gave feedback on these projects along the way, in particular colleagues at Facebook and Naver Labs. My colleagues in the Informatics Forum, in particular those from my cohort of the Data Science CDT (and everyone who helped run the cluster!) were always lovely to work with. I also couldn't have done this without support from my friends and family, in particular David Cooper, Ursula Stickland, Clare Cooper Stickland and Putri Viona Sari.

Declaration

I declare that this thesis was composed by me, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Asa Cooper Stickland*)

Contents

Chapter 1

Introduction

In this thesis we will tackle a problem in machine learning and natural language processing (NLP) that we will refer to as 'parameter efficient transfer learning'. At a high-level this involves taking 'general-purpose', large-scale models trained on huge amounts of text data from many domains, and specializing them to a particular task, while using as few resources as possible. The clearest examples of such models are language models/chatbots such as OpenAI's ChatGPT or Anthropic's Claude that can take a text input like 'write a poem in the style of William Wordsworth about the life of an Edinburgh PhD student' and produce a text output, like an example of such a poem.

Although products like ChatGPT offer the promise of giving users a helpful assistant that can solve almost any (sufficiently simple) task that takes text as input and output, empirically specializing a model to a particular settings (e.g. finance (Wu et al., 2023) or biomedical data (Zhang et al., 2023)) can outperform more general models. Additionally, models were significantly less useful as general-purpose assistants when the research for this PhD thesis was being done.

We can expect it will long be useful to create models that are specialized in some way, for example models specialized to:

- specific domains like medicine, law and computer programming.
- particular languages, especially 'low-resource' languages which are less represented in the training data of these general-purpose models.
- particular sources of input/information to the model like web search results, visual input, input from sensors/robots acting in the physical world, audio, and so on.

The motivation for designing cheaper methods for adaptation to these various settings is the huge scale of the models we are adapting, with huge storage costs associated with storing many copies of these models, and large amounts of GPU RAM and engineering efforts required to update model parameters, if done in a naive way. We tackle some of these questions of specializing to domain and language (and the combination of the two) in the later chapters of this thesis, and discuss the details of how to reduce the cost of adapting these models, and the trade-offs involved, in earlier chapters. A more detailed table of contents can be found in Section 1.1.

These large-scale, general-purpose models are of course a relatively recent phenomenon. Early machine learning systems often made use of hand-crafted features specific to particular tasks (e.g., SIFT (Lowe, 1999) in computer vision), avoiding the processing of raw data. Machine learning typically also relied on a 'supervised' learning approach which learned a mapping from an input (e.g. a movie review) to labels (e.g. positive/negative sentiment), which required labels manually

assigned by human labour. So called 'deep learning' systems, which can learn a multi-layered hierarchy of features and thus reduce the need for feature engineering, to a large extent replaced models using hand-crafted features, at least for tasks with sufficient amounts of labeled data. Although not a new idea, they became practical to train on large-scale tasks due to advances in hardware like better graphical processing units (GPUs) (Krizhevsky et al., 2012).

In this paradigm the choice of architecture and training setting for each task became the focus, meaning significant 'task-specific' work still had to be done. Initializing models with parameters resulting from training on large datasets was common in computer vision, mostly involving training on the large ImageNet (Deng et al., 2009) image classification dataset before adapting a model to some other task (Huh et al., 2016). However the most popular approach in NLP was to only initialise a small part of a model with already-trained parameters, for example vectors representing words (Collobert et al., 2011) that were fed into a neural network.

Recent advances have involved scaling up deep learning models for NLP, both in terms of the size of the dataset used to train these models, and the number of trainable parameters, i.e. the capacity of models to store information. This has led to the availability of powerful models that can be used for *many* tasks. These models are mostly based on a particular architecture, the 'transformer' (Vaswani et al., 2017). The initial training of these models is referred to as 'pre-training' due to this training process using an unsupervised objective (i.e. one that requires no human labels), and this thesis focuses on models trained on web-scale text corpora (for example, all of English Wikipedia or CommonCrawl[1]). Perhaps the canonical example is a transformer model known as BERT (Devlin et al., 2019), which was trained on English Wikipedia and Google Books, with an objective of filling in masked/blanked-out words in its input, e.g. given the input 'I _ to _ store' predict the words 'went' and 'the' if the original sentence was 'I went to the store'.

Although this 'masked language modeling' task is not obviously useful, by training on large-scale data, the model can learn information about the structure of English, and some amount of common sentence and world knowledge. We can use the resulting representation to achieve good performance when we fine-tune on a downstream task of interest, say, sentiment analysis (i.e. the task of classifying whether a review is positive or negative about a movie) with a small amount of supervised data. This paradigm has meant there is a lot of homogeneity in NLP: The same architecture and even specific model parameters are reused multiple times, motivating effort towards figuring out how to best use such models.

The typical pipeline is to 'fine-tune' these pre-trained models on a task of interest, and large-scale pre-trained models have achieved state-of-the-art results on many tasks this way. Such strong results are in part because of transferring knowledge from the pre-training corpus to the downstream task, and more generally using a model trained on one or more tasks (either supervised or unsupervised) that are different from your task of interest in order to get better performance on your task of interest is known as 'transfer learning'.

Transfer learning generally is broader and can encompass abilities like training on one language and generalizing to another language (without any additional training). Or it could refer to the ability of pre-trained autoregressive language models like GPT-3 (Brown et al., 2020) to perform tasks they were not directly trained on, simply by being trained for language modeling. However, in practice the most powerful forms of transfer learning involve the usual gradient updates to pre-trained models, such as the currently popular reinforcement learning from human feedback used in ChatGPT (Stiennon et al., 2020) or the aforementioned BERT fine-tuning.

Typically this fine-tuning of pre-trained models involves continuing training by updating every parameter of the pre-trained model on the new task. In this thesis we consider an alternative approach to fine-tuning every parameter where we only update a subset of (or small number of additional) pre-trained model parameters. This can save on computation, since we do not need

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

to calculate gradients for many parameters, and storage space, since we only need to store a potentially tiny 'update' to the large base model. This approach can also unlock new capabilities, like composing parameter-efficient modules trained on different tasks to each other, and in some situations outperform fine-tuning every parameter.

It's worth considering multiple definitions of 'efficiency'. First, 'memory efficiency' refers to the amount of computer memory (RAM, GPU memory, storage, etc.) required during both the training and inference phases of a model. Models that require less memory are desirable in resource-constrained environments and can enable more widespread use and easier deployment. We particularly focus on reducing memory costs during training. Second, 'sample efficiency' pertains to the speed at which models learn from new data. A sample efficient model requires fewer examples to generalize well to new tasks or domains, which is crucial when labeled data is scarce or expensive to obtain. Third, 'data efficiency' is closely related to sample efficiency but specifically focuses on the number of annotated data points necessary to achieve high performance (with no consideration for how many times we train on these data points). Because labeling data can be labor-intensive and costly, data efficient methods are sought after to minimize reliance on large annotated datasets. Lastly, 'parameter efficiency'—the focus of this thesis—deals with the number of parameters that must be modified or added to adapt a large-scale model to a particular task. These concepts can be closely related: for example if you have fewer parameters to learn, intuitively you might require fewer examples to learn from.

Early efforts in parameter-efficient transfer learning methods were designed for other domains, such as computer vision, and architectures, such as convolutional neural networks, and for smaller scale pre-trained models (Rebuffi et al., 2017, 2018; Swietojanski et al., 2016). One contribution of this thesis is the introduction of the parameter-efficient paradigm to the 'fine-tune a pre-trained transformer model' pipeline, with an initial exploration of where to insert task-specific parameters and which architectures to use.

In the first section we consider parameter-efficient transfer learning on English classification tasks, using the 'BERT' model mentioned above as a base model. Our first contribution is an approach to fine-tuning pre-trained models on multiple tasks simultaneously. Such multi-task learning can enable transfer between similar tasks, and efficiency by reducing the need for multiple task-specific models. However, typical approaches underperform task-specific models due to a lack of capacity and 'interference' between tasks. For example, the representations necessary for good performance on a sentiment analysis task might be different from those useful for a Natural Language Inference task, especially since pre-training has already given the model knowledge of basic syntax and semantics that we might hope to learn from the text data in various tasks.

Our contribution is an approach to multi-task learning where we introduce small task-specific modules for each task, which enable us to achieve the same performance as task-specific fine-tuning with only a fraction of the parameters. We also consider various ways to account for different tasks having big differences in the amount of training data they have, meaning we might undertrain on tasks with limited data.

This initial exploration was done on relatively small models compared to the current state-of-the-art, and did not cover a popular approach of freezing pre-trained model parameters and only training the small modules. In the second half of this section we address these limitations, contributing a survey of parameter-efficient approaches, showing which parameter-efficient architectures work the best as model scale increases, from 60 million to 3 billion parameters, and detailing tradeoffs between performance, memory-efficiency and other factors. We also draw connections from work in NLP to work in other domains, namely computer vision and speech, and also consider a complementary approach where the parameter-efficient modules' parameters are generated as a function of an input like a particular task or language.

In the second section we apply the parameter-efficient transfer learning approach to machine

translation (MT). This is a more complex task than classification. Consider translation from English to German. This involves modeling sequences (entire German sentences conditioned on their English translations) rather than class labels, and is multilingual rather than English-only. This means approaches designed for English classification can underperform. We explore adapting systems that have only been trained on an unsupervised objective (involving multilingual text but not machine translation) to the MT task. Specifically we study pre-trained transformers trained in a somewhat similar way to BERT, except the masked-out text comes in many languages and we use an architecture which can handle sequence generation, known as sequence-to-sequence models.

We were the first to apply parameter-efficient techniques to this latter problem, contributing several insights: We find that many more tunable parameters are needed for good performance in this setting compared to English classification, and we explore which parts of the transformer sequence-to-sequence architecture are important to adapt, e.g. finding that the cross-attention parameters are particularly important. We also explore using an English-only pre-trained model to improve translation performance.

An important problem in MT is domain adaptation, adapting a model from the translation of 'general' text to a specific style such as legal text, and we explore parameter-efficient approaches to this problem. This problem lends itself to the use of large pre-trained models, since the amount of domain-specific data will likely be quite small, motivating the need to start with a powerful base model to make up for the lack of in-domain signal. We tackle a more specific problem, cross-lingual transfer, where we want to generalize from domain-specific data in one pair of languages to another language pair where we don't have any data.

We contribute a new approach where we train independent 'adapters' (a particular parameter-efficient architecture) for source language, target language and domain, allowing us to compose them in ways not seen during training. For example, let's say during training we only saw data translating legal documents from French to English, and English to German. We can compose in a zero-shot fashion a French source adapter, a German target adapter, and a legal domain adapter to achieve better French to German translation despite never seeing this combination during training time.

There is significant overlap between relationship between parameter-efficient *transfer learning* and parameter-efficient *fine-tuning*. Full fine-tuning is probably the simplest form of transfer learning, but is surprisingly competitive with more complicated methods. Parameter-efficient fine-tuning simply aims to make this process cheaper and easier, and can be thought of as a drop-in replacement for normal fine-tuning, with the goal of exactly the same performance for less computational cost. 'Transfer learning' is broader and simply refers to getting training signal from one task in order to perform better on the another task. As mentioned in the previous paragraph, using parameter-efficient methods opens up new ways to do transfer learning, such as composing independent adapters. We also found parameter-efficient methods were useful for transfer learning in situations where regularization was especially important, such as using an English-only pre-trained model for machine translation.

As the final contribution of this theses, we conduct an extensive series of experiments, investigating empirically various questions. What parameter-efficient method architecture should we use? Are particular architectures better in terms of metrics like performance or memory usage? How does distance between source and target language pairs effect performance? And does the pre-trained model pre-training objective (can we train on an 'unsupervised' objective that doesn't rely on having the same sentence translated into different languages?) matter for good performance in machine translation when using parameter-efficient methods. We find that when translating distantly related language pairs like English and Korean the performance of parameter-efficient methods lags behind fine-tuning every parameter. We also find that for languages with limited training set sizes, parameter-efficient methods *outperform* full fine-tuning.

1.1 Structure of the thesis

The thesis consists of four main chapters (plus an introduction, background and conclusion). Three of these chapters are centered around published papers, Chapter 3 and Chapter 6 are centered around one paper and Chapter 5 is centered around two papers. Each chapter includes the paper(s) as published; in addition, it provides a little background in order to motivate the paper, and evaluates the contribution and impact of the paper since its publication. The chapters are as follows:

- Chapter 2 presents some basic concepts in machine learning and natural language processing (NLP), with a focus on the transformer architecture and 'pre-training' approach used prominently in this thesis.
- Chapter 3 presents the paper *BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning* (Stickland and Murray, 2019) which focuses on designing small, task-specific modules added on top of the BERT (Devlin et al., 2019) pre-trained model to avoid interference between tasks when fine-tuning BERT on multi-task learning.
- Chapter 4 is a survey of parameter-efficient methods, focusing on English classification tasks and the 'T5' pre-trained model family, discussing architecture choices, and trends in memory usage and performance as model scale increases.
- Chapter 5 presents the paper *Recipes for Adapting Pre-trained Monolingual and Multilingual Models to Machine Translation* (Stickland et al., 2021b), which investigates the benefits and drawbacks of freezing parameters, and adding adapters, when fine-tuning a pre-trained model on MT. We also present the paper *Multilingual Domain Adaptation for NMT: Decoupling Language and Domain Information with Adapters* (Cooper Stickland et al., 2021) which explores various aspects of parameter-efficient *domain adaptation* for machine translation, in particular focusing on using separate adapters for the source language, target language and domain in order to improve cross-lingual transfer from domain-specific data in one language pair to other language pairs.
- Chapter 6 presents the paper *When does Parameter-Efficient Transfer Learning Work for Machine Translation?* (Üstün and Cooper Stickland, 2022) in which we conduct extensive experiments, investigating to what extent parameter-efficient method architecture choice, distance between source and target language pairs, and pre-trained model objective matter for good performance in machine translation when using parameter-efficient methods.
- Chapter 7 concludes and provides a few directions for future research, focusing particularly on the relevance of our techniques to large decoder-only language models such as GPT-3 (Brown et al., 2020).

Chapter 2

Language models and pre-training

This chapter presents some basic concepts in machine learning and natural language processing (NLP), with a focus on the transformer architecture and 'pre-training' approach used prominently in this thesis. The reader might want to skip all or most of this introduction section if they are familiar with the transformer architecture and pre-training.

We start with the basics about representing text in ways that are useful for machine learning models (particularly neural networks) (section 2.2), most readers may want to skip over this. We assume some level of background in machine learning, for example knowing basic linear algebra, what a feed-forward network is, or knowing what a 'loss function' is. We recommend Murphy (2022) for an introduction to this material.

We then move on to a quick guide to the sequence-to-sequence approach to an NLP (section 2.2.3), where we also discuss the introduction of 'attention' mechanisms which later feature in the transformer architecture. For a more detailed introduction we recommend the tutorial by Neubig (2017) on neural machine translation, or Bahdanau et al. (2014) who introduced the attention mechanism.

Finally we give an overview of the transformer architecture, paying particular attention to the aspects important for our later discussion of parameter-efficient fine-tuning. We also discuss approaches for 'unsupervised' learning from large scale web corpora, known as 'pre-training' (section 2.3). For a more detailed transformer architecture explanation we recommend Elhage et al. (2021) who present a mathematical framework for thinking about what a transformer does, along with intuition for how it might implement various functions. We also recommend Vaswani et al. (2017) who introduced the architecture and an annotated version which includes PyTorch code snippets, `https://nlp.seas.harvard.edu/2018/04/03/attention.html`.

2.1 Notation

We use the following notation to denote vectors: $\vec{x}$, and capital letters and bold to indicate matrices, e.g. $\mathbf{X}$.

2.2 Modeling Text

Almost all of this thesis is concerned with neural network models that take text as input. Since neural networks take vectors of real numbers as input rather than words or characters, we need some way to transform text into a series of vectors. We are often also interested in 'representation

learning', or encoding text in such a way that it contains useful information or useful features. For a more comprehensive introduction we recommend Jurafsky and Martin (2023).

Imagine we have a training set consisting of various sentences we want to classify. Perhaps the most obvious way to represent this text is to break it up into words. The easiest way to represent a word as a vector is perhaps a one-hot vector, which has the dimension set to the vocabulary size V (the number of unique words in the training set), with an element of the vector corresponding to a particular word set to 1, and every other elements set to zero. We represent a word as a V-dimensional vector, with $x_i = 1$ for a word corresponding to the index i, and the other elements are zero. One-hot vectors contain no semantic information about words except simply distinguishing them from each other.

A simple extension of this idea allows us to represent a document as a bag-of-words (BOW) (Harris, 1954), where we use a single vector. This vector is equal to the sum of one-hot encodings of all words in the document: $\vec{d} = \sum_j \vec{x}^j$, where the jth word is represented by $\vec{x}^j$. Each entry d_i corresponds to the number of occurrences of the i-th word in the vocabulary in the document. In the standard BOW model, we consider only unigrams, sequences of one word. We may additionally consider bigrams, sequences of two words, trigrams, sequences of three words, and so on. However, the bag-of-words approach ignores word order, which can of course completely change the meaning of a sentence. There is also no way of encoding that two words might have similar meanings (e.g. 'happy' and 'ecstatic' would be represented by totally unrelated vectors).

We can avoid destroying such information by representing a sequence of L words $x_1, ..., x_L$ as a sequence of corresponding vector 'word embeddings', which can be thought of as columns of an 'embedding matrix' $\mathbf{M} \in \mathbb{R}^{V \times d}$, $M_{\cdot,x_1}, ..., M_{\cdot,x_L}$. See section 2.3 for early approaches to learning such word embeddings from data.

Another question is how to break up a piece of text into discrete pieces, a process known as 'tokenization', with the discrete pieces known as 'tokens'. One obvious way to do this is to separate it into individual words, and another is to separate into individual characters (e.g. 'hello' $\longrightarrow$ 'h e l l o'). Both approaches have downsides. Character level representations enable more graceful handling of typos and 'out-of-vocabulary' words that weren't seen at training time. However they lead to longer sequences and cannot encode much meaning on their own, requiring the context of surrounding characters to provide useful information.

Word level representations ensure each token is providing meaningful information and lead to shorter sequences, but are not able to handle out-of-vocabulary words, and cannot tell that e.g. 'swim', 'swimming' or indeed 'swwimming' are all related (at least without learning this from training data). Note also that simply splitting on whitespaces and ignoring numbers and punctuation will fail on cases like don't, P.h.D. or A&W. See Jurafsky and Martin (2023) section 2.4.1 for more details.

The approach used by most state-of-the-art models, and all models in this thesis, is known as 'subword' tokenization. Subword tokenization is a method of tokenization that involves breaking up words into smaller units, called subwords or word pieces, for example 'swimming' $\longrightarrow$ 'swim m ing'. This example shows the benefit of using subword tokenization, allowing the model to see that swimming is a modified version of the word 'swim', but not requiring the longer sequences of character-level representations. In order to use subword segmentation we need an algorithm to generate a vocabulary.

Byte Pair Encoding (BPE) is one such algorithm, first described publicly as a generic data compression algorithm by Philip Gage in a February 1994 article 'A New Algorithm for Data Compression' in the C Users Journal (Gage, 1994), and used for tokenization by Sennrich et al. (2016). It operates by iteratively replacing the most frequent pairs of characters in the training data with a single symbol (i.e. item in the vocabulary). This process is repeated until the desired vocabulary size is reached.

For example, suppose we have the following training data: 'the cat there, on the mat.' Using BPE, we might start by replacing the most frequent pair of characters, 'th', with a single symbol (we'll use '1'). This would result in the following: '1e cat 1ere, on 1e mat.' Next, we would replace the next most frequent pair of characters, 'at', with another symbol. This would result in the following data: '1e c2 1ere on 1e m2.' Now notice the pair of characters '1e' is the most common, so we can merge this into a single symbol, corresponding to 'the'. We would continue this process (over a larger training corpus) until we reached the desired vocabulary size.

The size of the vocabulary is a hyperparameter to be tuned, and picking a value too large or too small can result in representations too close to words for large values, or too close to characters for smaller values, but overall this process or close variants such as the SentencePiece (Kudo and Richardson, 2018) approach have become the workhorse of tokenization for large pre-trained models.

2.2.1 Autoregressive models

Often we are concerned with modelling the distribution of sequence of discrete random variables, i.e. a sequence of subword tokens. Autoregressive models are a general class of techniques that allow us to obtain a well-formed probability distribution over sequences of discrete random variables, such as subword tokens (Frey, 1998). In an autoregressive model, the probability distribution of the next token in a sequence is dependent on the previous tokens in the sequence. Mathematically, this can be expressed using the chain rule of probability. Let $x_1, x_2, ..., x_T$ be a sequence of subword tokens, where T is the length of the sequence. The probability distribution of this sequence can be expressed as:

$$p(x_1, x_2, ..., x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)...p(x_T|x_1, ..., x_{T-1}) = p(x_1)\prod_{t=2}^{T} p(x_t|x_1, ..., x_{t-1}) \tag{2.1}$$

This equation states that the probability of the entire sequence is the product of the probability of the first token and the probability of each subsequent token given the previous tokens in the sequence. In order to model sequences like this using deep learning, we need to ensure that x_t can only 'see' information from previous tokens, and not future ones, and we will see how to implement this in future sections. Autoregressive models can be used as generative models, by first sampling from $p(x_1)$, then using the result to sample from $p(x_2|x_1)$, and so on. One downside of this approach is that we have to run the model T times in order to sample an entire sequence. Autoregressive models can also give you the probability for a given input sequence, which some other generative models struggle with, for example the popular diffusion models used to generate images require extra computation to obtain probabilities.

Autoregressive models are useful for tasks such as language modeling, where the goal is to predict the next word in a sentence given the previous words. They can also be used for tasks such as machine translation, where the goal is to predict the next word in a sentence in the target language (the language we are translating into) given the previous words in the sentence and the entire source sentence (the sentence we are translating from).

2.2.2 Recurrent neural networks

An early neural network model of sequential input (which naturally gives us the property of current tokens being only able to access information from previous timesteps) was the recurrent neural network (Elman, 1990, RNN). An RNN can also be seen as a feed-forward neural network with a dynamic number of hidden layers that share the same parameters. Rather than applying a new transformation as we move up layers, the same transformation is applied for each new 'time step' in the sequence we are modeling. In contrast to a regular feed-forward neural network,

however, it accepts a new input at every 'layer' or time step. Specifically, the RNN maintains a hidden state $\vec{h}_t$, which represents its 'memory' of the contents of the sequence at each time step t. A simple RNN consists of several steps: We first 'look up' the vector $M_{\cdot,x_{t-1}}$ corresponding to the word x_{t-1}:

$$\vec{m}_t = M_{\cdot,x_{t-1}}, \tag{2.2}$$

leaving us with (independent) vector representations for each word at time t. We need to transfer information across timesteps to incorporate information from the whole sentence, and so the information at timestep t is stored in another vector, the 'hidden state' $\vec{h}_t$, which is updated using information from the input at the current timestep $\vec{m}_t$, as well as the hidden state from the previous timestep:

$$\vec{h}_t = \begin{cases} \tanh(\mathbf{W}_{mh}\vec{m}_t + \mathbf{W}_{hh}\vec{h}_{t-1} + \vec{b}_h) & t \geq 1, \\ \vec{0} & \text{otherwise.} \end{cases} \tag{2.3}$$

Note the weight matrices $\mathbf{W}_{hh}$ and $\mathbf{W}_{mh}$ are 'reused'/shared across every timestep. Finally we use the hidden state to make a prediction:

$$\vec{p}_t = \text{softmax}(\mathbf{W}_{hs}\vec{h}_t + b_s). \tag{2.4}$$

For the case of language modeling or a task where we classify each word separately such as part-of-speech tagging, this would correspond to choosing the probability distribution over the next word. For a sentence classification task, this would correspond to choosing the probability of the sentence label being a particular class, and we would only take the probability at the final timestep, i.e. after the RNN has 'seen' the entire sentence.

We've presented above perhaps the simplest possible RNN architecture, but we can add complexity in various ways. We can add additional 'layers', where the $\vec{m}_t$ vector comes from the previous layer hidden state instead of a word embedding. And many modifications to the simple update rule above have been proposed with the goal of aiding optimization and increasing the flexibility of the RNN architecture, with popular examples being the 'long short-term memory' or LSTM (Hochreiter and Schmidhuber, 1997), and 'gated recurrent unit' or GRU Chung et al. (2014). State-of-the-art results for RNNs typically used these more complicated architectures, but the details are unnecessary for the purposes of this thesis, since we use a different architecture entirely, see section 2.3.1.

2.2.3 Sequence-to-sequence models and attention

Many important tasks can be formulated as modeling the distribution of a sequence $\vec{y}$ given another sequence $\vec{x}$, $p(\vec{y}|\vec{x})$. For the case of machine translation, the 'source' $\vec{x}$ could be a German sentence and the 'target' $\vec{y}$ could be the English translation of the source sentence. Other $\vec{x}, \vec{y}$ examples could be a news article and its summary, or a piece of text and its paraphrase.

In general these tasks involve transforming one sequence into another, and so are known as sequence-to-sequence tasks. A popular family of models designed for these tasks are known as encoder-decoder models (Chrisman, 1991; Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014).

RNN Encoder-decoder Models

Before the advent of the 'transformer' architecture these encoder-decoder models were typically built from RNNs. The basic idea is relatively simple: we have an RNN language model to calculate the probabilities of the target $\vec{y}$, but we want to include information about the source sentence. A naive way to achieve this is to set the initial state of the language model as the output of another RNN that ran over the source sentence $\vec{x}$. The name 'encoder-decoder' comes

from the idea that the first neural network running over $\vec{x}$ 'encodes' its information as a vector of real-valued numbers (the hidden state), then the second neural network used to predict $\vec{y}$ 'decodes' this information into the target sentence. Figure 2.1 shows a schematic of this process. We will not cover the details of this approach since the rest of this thesis uses the 'transformer' architecture, which we will discuss in the next section, but the interested reader should consult Neubig (2017).

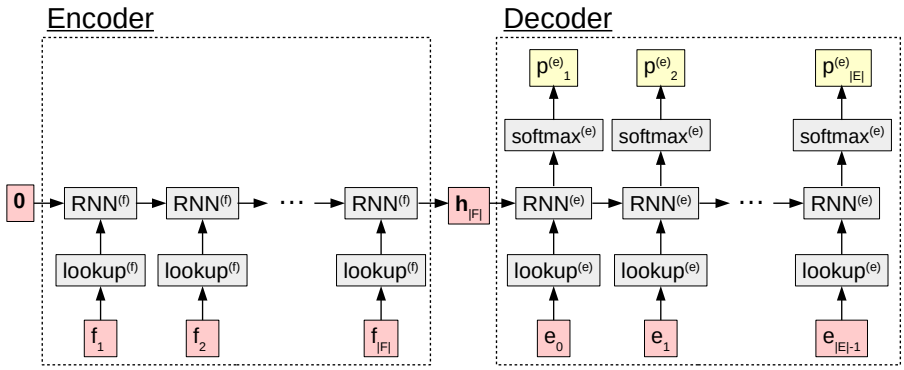


Figure 2.1: A computation graph of the encoder-decoder model. Figure from Neubig (2017).

2.3 Pre-trained Models

For many tasks we are interested in, it is impossible to collect a large, diverse training set. However if we are able to come up with an objective that doesn't require human labels (learning without labels is generally known as 'unsupervised learning'), we can use any freely available text to train our models. Given a diverse enough corpus, we may be able to learn things like syntax, world knowledge or even some kind of 'common sense'. For example training on Wikipedia should help with factual knowledge, or training on books might help with longer-form text.

There are many choices of unsupervised objective. Early approaches include Collobert et al. (2011), who learn word embeddings by training a neural network on a web corpus, with the objective of outputting a higher score for a correct word sequence than for an incorrect one (with the incorrect one formed by automatically corrupting the correct one). The 'word2vec' approach Mikolov et al. (2013), which was extremely popular before the advent of the transformer models discussed later in this section, uses the objective of predicting the words which appear in the context of a particular word. For example with the sentence 'I took swimming lessons', the vector embedding for the word 'swimming' would be trained to predict the word 'took' and 'lessons', assuming a context size of one.

Word vectors themselves are not particularly useful for understanding a sentence, and typically we need to do some processing on the entire sentence to understand it. A common pipeline was to start with pre-trained word embeddings and train a model such as a recurrent neural network that took embeddings as input and produced a representation of the entire sentence that can be used to solve a particular task.

Later approaches built on this idea, but instead of training a new RNN every time for a particular task, they 'pre-trained' an RNN with an unsupervised objective to produce 'contextual' embeddings for each word that took into account the context of surrounding words. For example Peters et al. (2018) trained an LSTM on a large web corpus with a 'bidirectional language modeling' objective. Here bidirectional means we both predict forwards in time, i.e. predict token $t + 1$ using the

previous t tokens, but also backwards in time, predicting token $t-1$ using the proceeding $L-t$ tokens where L is the length of the sentence.

These techniques are powerful but modern approaches almost all use a different neural architecture that seems to 'scale up' more gracefully, i.e. we can train much larger models on much more data and still squeeze out more performance. We discuss this architecture in the next section.

2.3.1 Transformers

A key bottleneck of recurrent models is their sequential nature. To compute the output for timestep t, we need to run the RNN on hidden states from timestep $t-1$, and to get *those* hidden states we need to run the RNN on hidden states from timestep $t-2$, and so on. At inference time this is not a problem, because autoregressive models require us to produce outputs one token at a time, no matter the details of their architecture. However it does create a bottleneck at training time.

Additionally, RNNs have some drawbacks, namely a lack of ability to deal with long-term dependencies and the need to store all the information about a sentence in the RNN hidden states. An 'attention' mechanism that can directly read information from many timesteps in the past is one way to alleviate these problems for RNN sequence-to-sequence models. Would it be possible to design an architecture entirely around such attention mechanisms?

The 'transformer' architecture introduced by Vaswani et al. (2017) is designed around 'attention' mechanisms. At a very high level, each layer of a transformer takes a sequence of vectors representing words in a sentence, and outputs another sequence of the same length. The input vectors are typically generated by an embedding layer that maps words in the input text to dense, continuous-valued vectors. Each layer is composed of a few 'sub-layers' or modules. These can be split into 'attention modules' based on the attention mechanisms we have already seen and feed-forward modules based on simple feed-forward networks (typically just a linear transform followed by a non-linearity, then another linear transform). The attention module deals with the relationship between tokens, for example using surrounding context of a particular word to understand e.g. what a pronoun is referring to, or disambiguate between two senses of the same word (e.g. 'I need to get money out of the bank' vs. 'I'm at the bank of the river'). The feed-forward module acts independently on every token, and empirically when the transformer is trained on large scale text from the web, neurons in these layers sometimes 'activate' for interpretable features like months of the year, specific famous people or even LaTeX commands (Elhage et al., 2022).

We'll now describe in detail these sub-layers, using the same notation that we will refer to in Chapter 4. We'll also highlight features and ideas relevant for parameter-efficient fine-tuning. This means assuming we have some 'pre-trained' model parameters that are frozen, and we want some lightweight mechanism to adapt this model to a new task. This might involve adding a small number of new parameters that we fine-tune, or only fine-tuning a tiny subset of existing parameters.

Self Attention Layer. Transformer attention modules are typically split into 'self-attention' and 'cross-attention', with the former designed to uncover relationships between a single sequence, and the latter designed to uncover relationships between two different sequences (such as the encoder output sequence and decoder sequence). We cover self-attention first. The input to a self-attention layer is a set of d-dimensional hidden states, one for each token in the input sequence (of length L), $\vec{h}_0, ..., \vec{h}_L$. In fact the attention mechanism of a transformer is known as 'multi-head attention' because multiple attention mechanisms are used at the same time. Multi-head attention performs the attention function in parallel over N_h heads, where each head (given the index e) is separately parameterized by $\mathbf{W}_q^e, \mathbf{W}_k^e, \mathbf{W}_v^e \in \mathbb{R}^{d_h \times d}$ to project inputs to 'queries', 'keys', and 'values'. d is the model dimension, and in MHA d_h is typically set to d/N_h

to avoid adding too many parameters, which means that each attention head is operating on a lower-dimensional space.

The basic attention mechanism used in this module looks like:

$$\vec{h}_j^e = \sum_{i=0}^{L} \text{softmax}((\mathbf{W}_k^e \vec{h}_i)^T (\mathbf{W}_q^e \vec{h}_j)) \mathbf{W}_v^e \vec{h}_i, \tag{2.5}$$

which we can write as

$$\vec{h}_j^e = \sum_{i=0}^{L} \alpha_{i,j}^e \mathbf{W}_v^e \vec{h}_i, \tag{2.6}$$

if we set $\alpha_{i,j}^e = \text{softmax}((\mathbf{W}_k^e \vec{h}_i)^T (\mathbf{W}_q^e \vec{h}_j))$. In this way we can see that the attention mechanism is giving us a weighted sum of value vectors. In other words the model can 'pay attention' to certain words or tokens by giving them a high weight. The language of queries, keys and values is meant to invoke concepts like a hashmap, where we 'compare' our query $\mathbf{W}_q^e \vec{h}_j$ to the key $\mathbf{W}_k^e \vec{h}_i$ using a dot product, and return the value $\mathbf{W}_v^e \vec{h}_i$ weighted by how similar the query and value are for index i.

Note that if we want to use these models for autoregressive language modeling (see section 2.2.1), we need to avoid using information from future timesteps when predicting the next word. In other words, when computing $p(\vec{x}_{j+1} | \vec{x}_{1:j})$, we need to ensure hidden states from timestep $j+1$ and greater do not influence hidden state j. This can be achieved by simply stopping the sum in Eq 2.6 at index j:

$$\vec{h}_j^e = \sum_{i=0}^{j} \alpha_{i,j}^e \mathbf{W}_v^e \vec{h}_i. \tag{2.7}$$

Returning to the idea of encoder-decoder models from section 2.2.3, we use the 'masked future' version of attention from Eq. 2.7 when using a transformer architecture in the decoder, and use the 'unmasked' version from Eq. 2.6 when using a transformer architecture in an encoder.

The previous transformations have left us with N_h output vectors, and we transform back to our original dimension with the following expression:

$$\vec{h}_j^o = \sum_{e=1}^{N_h} \mathbf{W}_o^e \vec{h}_j^e, \tag{2.8}$$

where $\mathbf{W}_o^e \in \mathbb{R}^{d \times d_h}$.

To make this more concrete, imagine we have the sentence "The capital of France is". In order to correctly predict the next token we need to output "Paris" (technically, we should put a high probability on that being the next token). One story we could tell is the key weight matrix $\mathbf{W}_k^e$ for this head 'searches' for cities, and $\mathbf{W}_q^e$ searches for the word 'is', such that the dot product $(\mathbf{W}_k^e \vec{h}_i)^T (\mathbf{W}_q^e \vec{h}_j)$ is only high when these two criteria are met, i.e. this dot product will only be high for the token "France". Then the softmax will ensure that the weight of this token is close to one, and the weight of the other tokens is close to zero, meaning the contribution of the value vectors $\mathbf{W}_v^e \vec{h}_i$ will also be close to zero except for at the France token.

Continuing our story, the value vector $\mathbf{W}_v^e \vec{h}_i$ followed by multiplication by the $\mathbf{W}_o^e$ matrix could be extracting the "capital city" concept from the "France" token, i.e. improving the prediction of the next word by increasing the probability of outputting Paris. In reality, the internal mechanisms might be much less clean than this, corresponding to many steps happening over multiple layers, but this is roughly the kind of computation attention mechanisms might be doing.

For the purposes of this thesis we normally consider the matrices $\mathbf{W}_q = \text{Concat}(\mathbf{W}_q^1 \cdots \mathbf{W}_q^{N_h}) \in \mathbb{R}^{d \times d}$ and $\mathbf{W}_o = \text{Concat}(\mathbf{W}_o^1 \cdots \mathbf{W}_o^{N_h})$ (with similar expressions for $\mathbf{W}_k$ and $\mathbf{W}_v$), since we 'adapt' these concatenated matrices in future chapters, and since most implementations of multi-head attention perform computation with these larger matrices for efficiency reasons.

Efficient fine-tuning. The 'parameter cost' of this layer is d^2 for each of $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$ and $\mathbf{W}_o$, leading to a total of $4d^2$. But note we could write the parameter cost of $\mathbf{W}_q$ as $N_h d_h d$, or 'number of heads' × 'head dimension' × 'model dimension'. So by reducing the number of heads N_h or the dimension of the lower-dimensional space d_h we can produce a lightweight version of this layer that can be chosen as the part of the network that we adapt, leaving the 'full-size' self-attention layer frozen, an idea we explore in Chapter 3.

A perhaps simpler idea is to freeze all of the individual weight matrices, but apply a 'low-rank correction' to them. This would look like taking the pre-trained query matrix $\mathbf{W}_q$ and replacing it with a new version $\mathbf{W}_q + \mathbf{U}\mathbf{V}^T$, where we only train the new matrix $\mathbf{U}\mathbf{V}^T$ and leave $\mathbf{W}_q$ frozen. Here $\mathbf{U} \in \mathbb{R}^{d \times r}$ and $\mathbf{V} \in \mathbb{R}^{d \times r}$ are smaller matrices since we assume $r << d$. This means they only contribute $2dr$ extra parameters, dr for each low-rank matrix. An alternative way to look at this is we are adding on a series of rank 1 corrections $\vec{u}_i \vec{v}_i^T$ leading to an overall rank r correction, i.e. $\mathbf{W}_q + \mathbf{U}\mathbf{V}^T = \mathbf{W}_q + \sum_{i=1}^{r} \vec{u}_i \vec{v}_i^T$. There is nothing special about $\mathbf{W}_q$ and we can in principle apply the same low-rank correction to any weight matrix in the transformer. See Chapter 4 for more exploration of this idea, known as 'LoRA'.

Finally, we can add a perturbation to the attention mechanism in Eq. 2.5. We replace the query ($\mathbf{W}_k^e \vec{h}_i$) and value ($\mathbf{W}_v^e \vec{h}_i$) vectors which depend on the input hidden state $\vec{h}_i$ with fixed 'virtual token' vectors of trainable parameters, and freeze all other parameters in the network. Again, see Chapter 4 for more exploration of this idea, known as 'Prefix Tuning'.

Cross Attention Layer. What if we also want to incorporate information from another sequence, typically the output of an encoder that we want to use at decoding time? Suppose we have the output of a transformer encoder (although any architecture that produces a set of vectors would be fine), another set of d-dimensional hidden states, one for each token in the input to the encoder (of length S), $\vec{y}_0, ..., \vec{y}_S$. This time we produce a weighted sum of encoder output states as follows:

$$\vec{h}_j^e = \sum_{i=0}^{S} \text{softmax}((\mathbf{W}_k^e \vec{y}_i)^T (\mathbf{W}_q^e \vec{h}_j)) \mathbf{W}_v^e \vec{y}_i, \tag{2.9}$$

so the jth hidden state $\vec{h}_j$ can incorporate information from any relevant encoder hidden state $\vec{y}_i$. We then add another transformation to bring us back to the same dimension as the decoder hidden states, in exactly the same way as Eq. 2.8.

Feed-Forward Layer. We apply a standard feed-forward network after the attention layers, i.e. a linear transformation followed by a non-linear activation function and another linear transformation. Importantly, the same transformation is applied to every element of the input sequence independent of the others, unlike the attention layer. For input matrix $\mathbf{X} \in \mathbb{R}^{L \times d}$ which represents packing the input hidden states $\vec{h}_i$ together, the output of the feed-forward layer is

$$\mathbf{F} = \text{FFN}(\mathbf{X}) = \sigma(\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{2.10}$$

where $\sigma(\cdot)$ is the activation function, and $\mathbf{W}_1 \in \mathbb{R}^{d \times d_f}$, $\mathbf{b}_1 \in \mathbb{R}^{d_f}$, $\mathbf{W}_2 \in \mathbb{R}^{d_f \times d}$, $\mathbf{b}_2 \in \mathbb{R}^d$ are all learnable parameters.

One way to view this layer is as a key-value store, where the keys and values are 'concepts' that we search for, and add to, the input hidden state respectively. For example we might search for the concept of 'Eiffel' in the input and reinforce the concept of 'France'. We can rewrite the layer as a series of vectors $\vec{g}_j$ (the jth row of $\mathbf{W}_2$) being added to the input hidden states, with

the weight given to these vectors being determined by the dot product of the hidden state with another series of vectors $\vec{f_j}$ (the jth column of $\mathbf{W}_1$). This would look like

$$\vec{h}_i' = \sum_{j=1}^{d_f} \sigma(\vec{f}_j^T \vec{h}_i)\vec{g}_j. \tag{2.11}$$

And so for a given concept, the dot product or overlap between the two vectors $\vec{f}_j^T \vec{h}_i$ represents the strength of the concept in the input hidden state $\vec{h}_i$. This leads to the output $\vec{h}_i'$ being a combination of various concepts encoded in the vectors $\vec{g}_j$, weighted by the non-linearity $\sigma(\cdot)$ applied to dot product. As we will see in the next few paragraphs, this output is added on to a copy of the input, so we can think of it as re-weighting concepts from the input hidden state. Note this is only an analogy, explored in Geva et al. (2021), and we don't have a full picture of what kinds of computation occurs in the feed-forward layers; for example, some vectors don't cleanly map onto concepts.

Efficient fine-tuning. Typically the intermediate representations are larger than the hidden dimension, i.e. $d < d_f$, and in particular usually $d_f = 4d$ (this was the case for e.g. BERT (Devlin et al., 2019) and GPT-3 (Brown et al., 2020)). This means the parameter cost of this layer is higher than the self-attention layer, namely (ignoring bias parameters) $dd_f = 4d^2$ from $\mathbf{W}_1$ and $d_f d = 4d^2$ from $\mathbf{W}_2$, for a total of $8d^2$ parameters, as opposed to $4d^2$ from the self-attention layer. Suppose again that we have frozen most model parameters and want to tune only a small number of them on some downstream task. A natural thought might be to create a lightweight version of this layer, for example one where $d_f << d$, and treat it as an additional sub-layer of the full transformer layer. We explore this idea in Chapter 3 and Chapter 4.

Residual Connection and Normalization. Residual connections and normalization layers are used in almost all Transformer models. Typically normalisation is either applied after the residual connection (post-norm), or before the sub-layer (pre-norm). Pre-norm is used by the 'T5' model we consider in Chapter 4, i.e. we apply normalisation before the layer $f(\cdot)$, and add a residual connection to the input as follows,

$$\text{NormResidual}(\mathbf{X}, f) = \mathbf{X} + f(\text{Norm}(\mathbf{X})), \tag{2.12}$$

where $\text{Norm}(\cdot)$ denotes the normalization operation, typically 'layer-norm' Ba et al. (2016), which sets the mean of a hidden state to zero and the variance to one. Post-norm transformer variants would replace this operation with the following:

$$\text{Add\&Norm}(\mathbf{X}, f) = \text{Norm}(\mathbf{X} + f(\mathbf{X})). \tag{2.13}$$

This is the variant to used by the BERT model. We show a schematic diagram of this post-norm variant for an encoder-decoder transformer in Figure 2.2.

Because of the residual connection, information is always passed directly from the input to the output of each sub-layer, and we can think of each sub-layer as 'perturbing' the input. This might involve 'up-weighting' concepts as we discussed previously when introducing the feed-forward layer, or taking information from one token and adding it to the representations in another token as might be typical from a self-attention layer. This idea is sometimes referred to as the 'residual stream' (Elhage et al., 2021), e.g. "the 'water' concept was extracted from the residual stream of the 'river' token, and added to the residual stream of the 'bank' token."

Full (Encoder) Transformer Layer. We can express the transformer layer in quite a simple form, essentially applying the self-attention module **MHA** and then the feed forward module **F**, with residual connections and normalization in between:

$$\begin{aligned} \mathbf{Y} &= \text{NormResidual}(\mathbf{X}, \mathbf{MHA}) \\ \mathbf{Z} &= \text{NormResidual}(\mathbf{Y}, \mathbf{F}), \end{aligned} \tag{2.14}$$

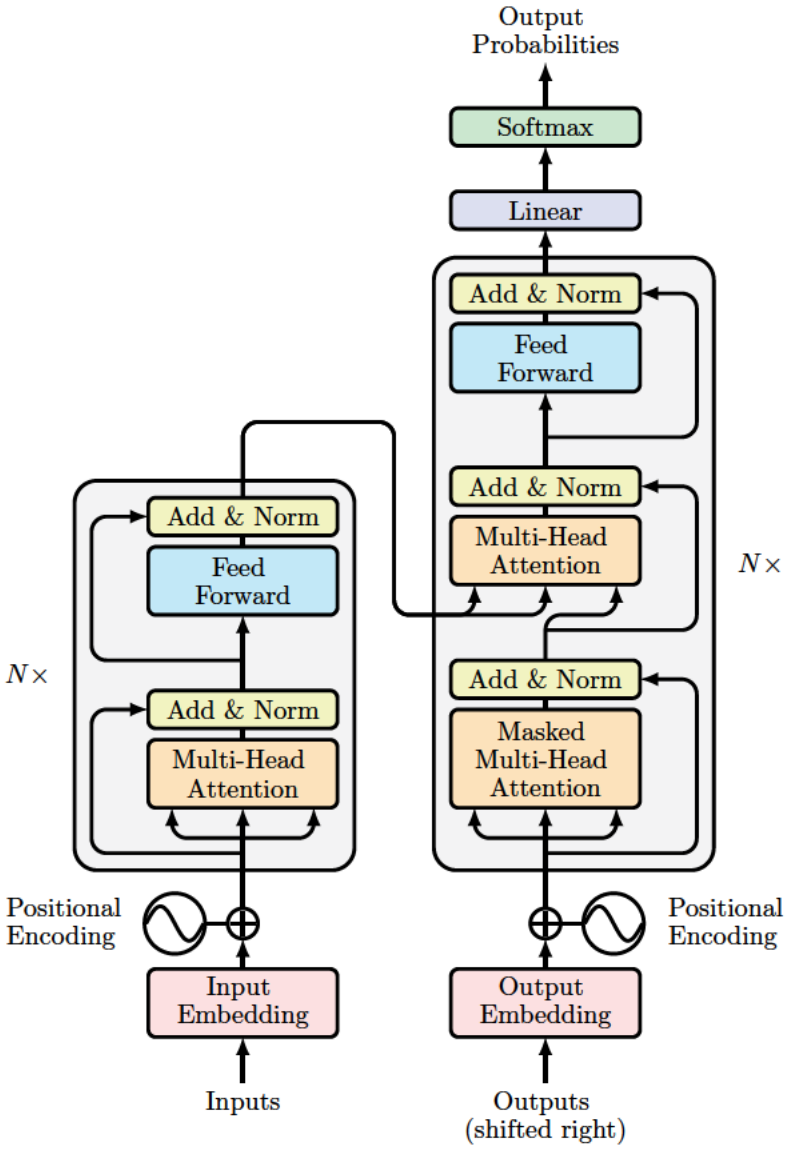


Figure 2.2: Schematic diagram showing the post-norm, encoder-decoder transformer variant.

where $\mathbf{Y}$ is the intermediate representation after the attention block, and $\mathbf{Z}$ is the output of the layer.

The full transformer model is built from stacking several of these layers, with the following additional steps. First, an embedding lookup table to transform from tokens to vectors that can be an input to the transformer layers. Second, vectors that encode the position of each token in the sequence, and finally some output module that transforms from the final hidden states of the final layer into predictions e.g. a probability distribution over the next word in the sequence if we are performing language modeling, or the probability distribution over labels assigned to the sequence in the case of classification. A standard transformer decoder-only architecture designed for language modeling would incorporate a mask that doesn't allow information from timestep t to flow to hidden states or activations from timestep $< t$ in the self-attention module.

Full (Decoder) Transformer Layer. A layer from a decoder in a standard transformer encoder-decoder architecture would incorporate this mask, and additionally there would be a *cross-attention* module after the self-attention one, taking as input both $\mathbf{Y}$ and the output of a transformer encoder $\mathbf{E}$. We again apply the self-attention module **MHA** and then the feed forward module $\mathbf{F}$, but additionally include a cross-attention layer **CA** where the 'query' vectors come from the hidden states of the decoder $\mathbf{Y}_1$, again with residual connections and normalization in between:

$$\begin{aligned} \mathbf{Y}_1 &= \text{NormResidual}(\mathbf{X}, \mathbf{MHA}) \\ \mathbf{Y}_2 &= \text{NormResidual}(\mathbf{Y}_1, \mathbf{CA}) \\ \mathbf{Z} &= \text{NormResidual}(\mathbf{Y}_2, \mathbf{F}), \end{aligned} \tag{2.15}$$

where $\mathbf{Y}_2$ is the intermediate representation after the two different attention blocks, and $\mathbf{Z}$ is the output of the layer.

Since we've only given enough detail to give the reader a rough idea of how the transformer architecture works, we recommend the following resources for obtaining a deeper understanding: Elhage et al. (2021) presents a mathematical framework for thinking about what a transformer does, along with intuition for how it might implement various functions (such as increasing the probability of 'Potter' appearing after the word 'Harry' when there are many previous occurrences of the bigram 'Harry Potter' in a document - A concept known as 'induction heads'). We also recommend Vaswani et al. (2017) who introduced the architecture and an annotated version which includes PyTorch code snippets, `https://nlp.seas.harvard.edu/2018/04/03/attention.html`.

2.3.2 Pre-trained Language Models

Combining this transformer architecture with large scale pre-training with an unsupervised objective on data scraped from the web has proven to be a powerful recipe in NLP, with many state-of-the-art results coming from this setup. In this section we describe two strategies for pre-training that are used to obtain the models we use in this thesis. Another strategy, using a simple auto-regressive language modeling objective, has become increasingly popular (Brown et al., 2020; Chowdhery et al., 2022; Rae et al., 2021), but due to our focus on classification tasks where other architectures empirically perform better (Tay et al., 2022b), and machine translation where we focus on models that are multilingual and have encoder-decoder architectures, we mostly ignore large decoder-only language models. We discuss this further when giving our thoughts on possible future work in Chapter 7.

Masked Language Modeling. Many of the first pre-trained models based on the transformer architecture to become popular were encoder-only architectures, i.e. they did not use autoregressive masking in the attention mechanism, allowing every token to 'see' every other token. The BERT (Devlin et al., 2019) model was the first to do this kind of large-scale pre-training with an encoder-only architecture. It is pre-trained with a 'masked language modeling' (MLM) task

and a 'next sentence prediction' (NSP) objective. During the pre-training process, the input to the model is a pair of sequences, with special tokens added to indicate the beginning and end of the sequences. Some of the tokens in the input sequences are also replaced with `[MASK]` tokens, which are used to remove information about these tokens from the model. We use the BERT model in Chapter 3, for English text classification tasks.

The goal of the MLM task is to maximize the conditional probability of the label tokens at the `[MASK]` positions in the input sequences. This is done by training the model to predict the original token at the `[MASK]` position given the rest of the input sequence. The loss for the MLM task is calculated as the negative log probability of the original token at the `[MASK]` position, as shown here:

$$\mathcal{L}_{\text{MLM}} = -\sum_{x_m \in M(\mathbf{x})} \log \mathbf{P}(x_m | \mathbf{x}_{\setminus M(x)}), \tag{2.16}$$

where $M(\mathbf{x})$ refers to the set of tokens that were masked.

In addition to the MLM task, the BERT model is also pre-trained with a next sentence prediction (NSP) task, which involves predicting whether two input sequences are coherent. The final representation of the special `[CLS]` token is used for this prediction. Pre-training the model on these tasks means the model should learn features useful for predicting the masked words, giving the model the ability to contextualize the words in an input sentence. Various improvements have been made to this basic recipe such as training for longer and without the NSP objective Liu et al. (2019) or replacing the masking objective with an objective of classifying which input tokens had been replaced with the outputs of a masked language model (this second model is trained alongside the first with the standard MLM objective) (Clark et al., 2020).

Denoising Auto-encoder. While masked language modeling is suited to obtaining models that do well on classification tasks, it lacks an obvious way to generate text for the tasks that require it like translation or summarization (it is possible to sample from models trained with the MLM objective but empirically this strategy does not perform as well as the one we describe next). Many tasks can be framed as conditional sequence modeling (e.g. model target sentences conditional on source sentences for machine translation), making an encoder-decoder architecture ideal. The encoder is free to remove any masks from the attention mechanism since the target is conditional on the entire source/input sentence and we don't need to sample any source sentences, while the decoder can use an autoregressive masking strategy to enable sampling of the target sentence.

Such encoder-decoder architectures are often trained with so-called 'denoising autoencoder' objectives, where we take a corrupted version of some input and try and recover the original 'clean' input. The most popular models using this strategy, and the ones we mostly use in this thesis, are the following: Firstly BART (Lewis et al., 2020) (and various multilingual versions of it which we will cover in Chapter 5). The basic idea is to take a sentence or document $\mathbf{x}$, apply some noise to it (we assume some 'noising' in function $N(\mathbf{x})$), then train the model on the task of recreating the original sentence given the noisy version, or more precisely optimizing the following objective:

$$\mathcal{L}_{\text{AE}} = -\sum_{t} \log \mathbf{P}(x_t | \mathbf{x}_{1:t-1}, N(\mathbf{x})). \tag{2.17}$$

BART specifically using a noising function that masks 30% of tokens in each document, and uses 'sentence permutation', where a document is divided into sentences based on full stops, and these sentences are shuffled in a random order. In principle any noise or corruption can be applied to the input, making this strategy more flexible than the masked language modeling one. BART was trained on English data, and achieved strong results at the time at tasks like news summarization. We focus mostly on using sequence-to-sequence models for machine translation,

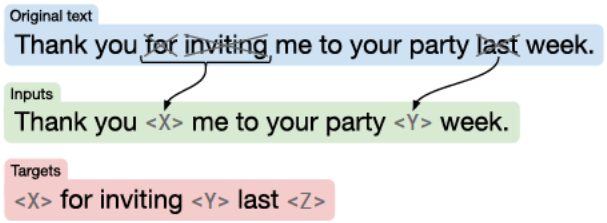


Figure 2.3: An example of the T5 objective from Raffel et al. (2019). The 'target' corresponds to $\mathbf{y}$, and 'inputs' correspond to $\mathbf{x}_{\setminus M(x)}$ in equation 2.18.

meaning English-only models are not ideal, and we cover the difference between starting from a multilingual model vs an English-only one in Chapter 5.

We also use T5 (Raffel et al., 2019), which uses an objective that specifically corrupts contiguous, randomly spaced spans of tokens (this is very similar to the MLM corruption strategy, except for in T5 we corrupt *spans* of tokens rather than sampling tokens to mask i.i.d.). All consecutive spans of dropped-out tokens are replaced by a single sentinel token. Each sentinel token is assigned a token ID that is unique to the sequence. The target sentence then corresponds to all of the dropped-out spans of tokens, delimited by the same sentinel tokens used in the input sequence plus a final sentinel token to mark the end of the target sequence; see Figure 2.3 for a visual representation. Thus the objective looks like a 'encoder-decoder version' of the MLM objective, with a similar masking strategy, but predicting the masked tokens in the decoder rather than the encoder. We can write the objective out as the following,

$$\mathcal{L}_{\text{T5}} = -\sum_{t} \log \mathbf{P}(y_t | \mathbf{y}_{1:t-1}, \mathbf{x}_{\setminus M(x)}), \tag{2.18}$$

where $M(\mathbf{x})$ refers to the spans of tokens that were masked, and $\mathbf{y}$ consists of every corrupted token (or every element of $M(\mathbf{x})$), ordered so that they come in the same order as in the uncorrupted sequence. We use this model in Chapter 4, due to its strong performance on challenging English classification tasks and the availability of T5 models of various sizes ranging from hundreds of millions to billions of parameters, allowing us to see how our techniques scale with pre-trained model size.

2.4 Conclusion

We now have an understanding of how we might create useful representations within models by pre-training them on large amounts of text, and have seen the different elements of the transformer architecture. Along the way we tried to highlight how we might modify the typical transformer architecture such that we can fine-tune only a small number of parameters on some downstream task. The next chapter focuses on our work released shortly after BERT that concentrated on parameter-efficient methods applied to multi-task learning, and Chapter 4 surveys a range of methods that have been proposed since.

Part I

Parameter-efficient transfer learning

Chapter 3

BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning

3.1 The paper

This section presents the paper *BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning* (Stickland and Murray, 2019).

The paper was initially published as a preprint on arXiv in January 2019. Then, it was accepted for publication at the conference *International Conference on Machine Learning (ICML)* in June 2019.

Along with concurrent work (Houlsby et al., 2019) this paper was the first to apply 'parameter efficient' methods to NLP, and pre-trained transformers in particular. We concentrate on multi-task learning on 7 of the 8 tasks in the GLUE benchmark, starting from a pre-trained transformer encoder, BERT. We avoid 'interference' between tasks by adding lightweight, task-specific modules at every layer and recover the same performance as training an entire network on each task separately while doing multi-task learning. We explore various strategies for how to add parameters to a pre-trained model, exploring tradeoffs between sharing representations between tasks by only adapting 'higher' layers and maximizing performance by adapting every layer.

Our main contributions were the following:

- Introducing 'Projected Attention Layers' (PALs), lightweight versions of transformer attention layers.
- Introducing 'annealed sampling' in which during multi-task learning we start by sampling tasks proportional to dataset size but sample tasks equally at the end of training, linearly interpolated between these two over the course of training.
- Bringing the framework of Rebuffi et al. (2018) from computer vision to NLP (and from convolutional neural networks to transformers), by using low-rank 'adapter' modules inserted after every layer of a pre-trained model.

And our main findings were:

- We could match the performance of fine-tuning a separate network for each task by using multi-task learning with a small number of task-specific parameters (either low-rank layers or PALs). For one dataset, RTE, we improved performance over standard single-task fine-tuning, with the amount of positive transfer increasing with more shared parameters (not using task-specific parameters led to more transfer).
- Adding parameters 'on top' of the existing network did not help very much, but adding parameters (PALs) to the top six layers outperformed adding parameters to the bottom six layers.
- Annealed sampling gave the best performance, and sampling tasks proportional to the square root of dataset size also worked better than simply sampling proportional to dataset size or sampling equally.

Author contributions

Asa Cooper Stickland came up with the initial idea, wrote the code for and carried out all experiments, and largely wrote the paper.

Iain Murray provided feedback on which experiments to carry out, guided the focus and scope of the paper, and provided detailed comments on the paper and re-wrote some sections.

BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning

Asa Cooper Stickland [1] **Iain Murray** [1]

Abstract

Multi-task learning shares information between related tasks, sometimes reducing the number of parameters required. State-of-the-art results across multiple natural language understanding tasks in the GLUE benchmark have previously used transfer from a single large task: unsupervised pre-training with BERT, where a separate BERT model was fine-tuned for each task. We explore multi-task approaches that share a single BERT model with a small number of additional task-specific parameters. Using new adaptation modules, PALs or 'projected attention layers', we match the performance of separately fine-tuned models on the GLUE benchmark with ≈7 times fewer parameters, and obtain state-of-the-art results on the Recognizing Textual Entailment dataset.

1. Introduction

This work explores how to adapt a single large base model to work with multiple tasks. In particular we focus on using deep neural networks, pre-trained on large amounts of English text, for multi-task learning on several natural language understanding (NLU) tasks.

Some multi-task learning approaches consider learning a general-purpose model that shares all parameters across tasks (e.g., the NLP decathlon introduced by McCann et al., 2018). This setting requires all tasks to have the same input and output space, and the input indicates the task. Instead, we consider the setting where we share most parameters across all tasks, but have a small number of task-specific parameters which adapt the shared model.

Sharing parameters, and thus a common representation, between tasks can sometimes lead to better generalization. However, fine-tuning separate models for each task often works better in practice. Although we are interested in multi-task methods that give results close to (or better than) state-of-the-art, there are separate motivations for maintaining shared parameters between tasks:

- On applications like mobile devices we may have constraints on battery life. Applying several different neural networks to the same input costs energy. If only the 'tops' of our models are task-specific, we can apply a shared transformation only once to the input, and use this transformed representation multiple times, as input to each task-specific function.
- Again on mobile devices, running several different neural networks for various tasks can incur a computational and energy overhead due to swapping parameters on a dedicated integrated circuit (Rebuffi et al., 2018).
- An application with a large number of tasks may have constraints on the number of parameters that can be stored. For example, web-scale applications may need to avoid storing a separate large model for every user.

Given a large number of shared parameters in a base model, and a small number of task-specific parameters, our key questions are: where should we be transforming the base model? What form should these transformations take? We assume the task is always known, so the model can always choose the correct adaptation parameters and output space.

We experiment on a set of eight NLU tasks from the GLUE benchmark (Wang et al., 2018a), which include question answering, sentiment analysis, and textual entailment. The number of training examples varies widely across the tasks, so we explore how to schedule training to not unduly favor the well-resourced tasks, or overfit the low-resource tasks.

We use the BERT model (Bidirectional Encoder Representations from Transformers, Devlin et al., 2018) as our base pre-trained model. Pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, including the GLUE benchmark. However, the entire model is fine-tuned, meaning we need a separate model for each task. The

[1]School of Informatics, University of Edinburgh. Correspondence to: Asa Cooper Stickland < >.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

transformer architecture that BERT is based on is powerful and popular, so finding the best way to adapt the parameters of this architecture for multi-task learning may be useful in other contexts, such as multilingual machine translation.

Our main contributions are: 1) We introduce the 'Projected Attention Layer' (PAL), a low-dimensional multi-head attention layer that is added in parallel to normal BERT layers. 2) We introduce a novel method for scheduling training, where we sample tasks proportional to their training set size at first, and de-emphasize training set size as training proceeds. 3) We perform an empirical comparison of alternative adaptation modules for self-attention-based architectures.

Making links to the vision literature, we identify shared lessons for where to add task-adaptation parameters depending on resource constraints. On the GLUE benchmark, we show that PALs enable comparable performance to fine-tuned BERT-base (the smaller of the two models considered by Devlin et al. 2018) on many tasks with ≈7 times fewer parameters. We improve the performance of BERT-base on the recognising textual entailment (RTE) task, achieving 76.6% accuracy, surpassing the performance of fine-tuned BERT-large (70.1%) and the MT-DNN model (Liu et al., 2019) (75.5%) which also uses BERT and multi-task learning. We also find that the more parameter sharing we have, the better we do on the RTE task.

2. Background

Multi-task learning aims to provide an inductive bias that means models have to learn features that are general enough to perform well on many tasks (Caruana, 1997). In NLP, examples of previous work include using a single model for chunking, tagging, named entity recognition, and semantic role labeling by applying a shared neural network to text, with different output layers (Collobert et al., 2011). Another approach outputs predictions at different layers using the idea of a linguistic hierarchy (Hashimoto et al., 2017; Sanh et al., 2018). Subramanian et al. (2018) train a sequence-to-sequence RNN model on tasks including machine translation and natural language inference, and learn sentence representations useful for downstream tasks. Outside NLP, multi-task learning has been applied to diverse domains such as speech recognition (Deng et al., 2013) and reinforcement learning (Teh et al., 2017). Ruder (2017) provides a more general overview.

Many multi-task learning approaches can be categorized as either 'hard parameter sharing' or 'soft parameter sharing'. Hard parameter sharing uses the same hidden layers for all tasks, with task-specific output layers. Soft parameter sharing gives each task its own model, but the distances between the parameters of the models are regularized to encourage the parameters to be similar. For example Duong et al. (2015) use the L2 distance, and Yang & Hospedales (2017) use the trace norm. In this work we assume that soft-parameter sharing with the whole of BERT requires too many parameters. We instead explore how to do hard-parameter sharing, by adding adapters to shared layers, as well as the usual separate output layers.

2.1. Adaptation Parameters

Various strategies for adding adaptation parameters have been explored. *Learning hidden unit contributions* (LHUC, Swietojanski & Renals, 2014) modifies a neural network by multiplying each hidden unit by a learnable scalar. Since the number of units is much smaller than the number of parameters in the network, this approach adds a small number of parameters compared to other methods we consider.

Residual adapter modules (Rebuffi et al., 2018) adapt large pre-trained residual networks (He et al., 2016) for multi-task learning in computer vision. Each adapter module contains a 1×1 filter bank with a skip connection, which can be inserted *in series*, between the original network layers, or *in parallel*, as additional inputs to a layer. For a layer with C channels, the module contains an additional $C\times C$ matrix per layer for each task, containing C 1×1 convolutional filters. This $C\times C$ matrix can be compressed by replacing it with a low-rank approximation, so that the adapters contain a small fraction of the model parameters (e.g., less than 10% for each task). Several of our methods were inspired by the idea of using a low-rank approximation to the key operation of a model: the convolutional layer when dealing with images, or multi-head attention in the transformer.

2.2. Fine-tuning Approaches

A recent trend in transfer learning is to pre-train some model architecture on a language modeling objective before fine-tuning that same model for a supervised downstream task (Dai & Le, 2015; Howard & Ruder, 2018; Radford, 2018). BERT uses a similar approach, but was pre-trained with two objectives: 1) filling in words 'masked' out of an input sentence, and 2) classifying whether two input sentences are adjacent in a corpus of text. Unlike a normal language modeling objective, BERT conditions on both left and right context when predicting the masked words.

The neural network layers in BERT are taken from the Transformer model (Vaswani et al., 2017), a sequence to sequence model that achieved state-of-the-art results in machine translation. Transformer layers have subsequently been used more broadly, e.g. for language modeling (Dai et al., 2019), image generation (Zhang et al., 2018), and generalized to video classification, object detection/segmentation and human pose estimation (Wang et al., 2018b).

A concurrent approach by Houlsby et al. (2019), introduces

adapters similar to our 'low-rank' layers (section 3.3), but added within each layer before each application of layer-norm. This work also keeps the BERT model fixed while training adapter modules. We concentrated on jointly fine-tuning the entire BERT model on all tasks, which has downsides: 1) interference and 'forgetting' of stored knowledge is possible; 2) we require access to all tasks at training time. However the multi-task setup requires less adaptation parameters for good performance (we use $1.13\times$ parameters compared to their $1.3\times$ parameters[1] to match having separate models for each GLUE task.), and is crucial for the transfer effects that gave us good performance on RTE.

3. Adapting Self Attention

The BERT model we are adapting is a multi-layer bidirectional Transformer encoder based on the original model of Vaswani et al. (2017). We only consider the smaller BERT-base model, which contains 110 million parameters. We somewhat arbitrarily limit ourselves to a $1.13\times$ increase in total parameters, which is equivalent to 15 million, or 1.9 million parameters per task. This choice avoids the extremes of having nearly no extra task-specific parameters, or giving each task its own whole model.

In the following sections we first introduce various components of the full BERT model, and discuss how many parameters they require (section 3.1). We then show the exact form our parameter additions took, distinguishing between adding to the 'top' of the model, just before the output space (section 3.2), or within each layer of the BERT-base architecture (section 3.3).

3.1. Model Architecture and Multi-head Attention

BERT takes in a sequence (one or two English sentences in our case) and outputs a vector representation of that sequence. Each token in the sequence has its own hidden vector, and the first token of every sequence is always a special classification embedding (`[CLS]`). At each layer of BERT the hidden states of every sequence element are transformed, but only the final hidden state of `[CLS]` is used for classification/regression tasks. We now describe how the vector for one element of the sequence is transformed.

The multi-head attention layer (Vaswani et al., 2017) is the core of the transformer architecture that transforms hidden states for each element of a sequence based on the other elements (the fully-connected layers act on each element separately). The multi-head layer, which we write as $\mathrm{MH}(\cdot)$, consists of n different dot-product attention mechanisms. At a high level, attention represents a sequence element with a weighted sum of the hidden states of all the sequence elements. In multi-head attention the weights in the sum use dot product similarity between transformed hidden states.

Concretely, the ith attention mechanism 'head' is:

$$\mathrm{Attention}_i(\mathbf{h}_j) = \sum_t \mathrm{softmax}\left(\frac{W_i^q \mathbf{h}_j \cdot W_i^k \mathbf{h}_t}{\sqrt{d/n}}\right) W_i^v \mathbf{h}_t \tag{1}$$

where $\mathbf{h}_j$ (we drop the j index in the following discussion) is a d dimensional hidden vector for a particular sequence element, and t runs over every sequence element. In BERT the W_i^q, W_i^k and W_i^v are matrices of size $d/n \times d$, and so each 'head' projects down to a different subspace of size d/n, attending to different information. Finally the outputs of the n attention heads (each of size d/n) are concatenated together (which we show as $[\cdot, ..., \cdot]$) and linearly transformed:

$$\mathrm{MH}(\mathbf{h}) = W^o \left[\mathrm{Attention}_1(\mathbf{h}), ..., \mathrm{Attention}_n(\mathbf{h})\right] \tag{2}$$

with W^o a $d \times d$ matrix[2]. Throughout this section, we ignore terms linear in d (like bias terms) to avoid clutter, as they don't add significantly to the parameter count. The matrices in a multi-head layer have $3nd^2/n + d^2 = 4d^2$ parameters.

We further define another component of a BERT layer, the self-attention layer, which we write as $\mathrm{SA}(\cdot)$:

$$\mathrm{SA}(\mathbf{h}) = \mathrm{FFN}(\mathrm{LN}(\mathbf{h} + \mathrm{MH}(\mathbf{h}))), \tag{3}$$

$\mathrm{LN}(\cdot)$ is *layer normalisation* (Ba et al., 2016), requiring $2d$ parameters. FFN is a standard *feed-forward network*,

$$\mathrm{FFN}(\mathbf{h}) = W_2 f(W_1 \mathbf{h} + b_1) + b_2, \tag{4}$$

with $f(\cdot)$ a non-linearity, GeLU (Hendrycks & Gimpel, 2016) in BERT. Matrix W_1 has size $d_{ff} \times d$ and W_2 has size $d \times d_{ff}$, so overall we require $2dd_{ff}$ parameters from the FFN component.

Putting this together, a BERT layer, which we write $\mathrm{BL}(\cdot)$, is layer-norm applied to the output of a self-attention layer, with a residual connection.

$$\mathrm{BL}(\mathbf{h}) = \mathrm{LN}(\mathbf{h} + \mathrm{SA}(\mathbf{h})) \tag{5}$$

We have $4d^2 + 2dd_{ff}$ total parameters from a BERT layer.

The entire BERT model is simply a stack of 12 BERT layers, followed by (in our case) a transformation to take us to the output space for a NLU task. We write the dimensions of the hidden states in BERT-base as $d_m = 768$. The final hidden state of the first token of every sequence is all that is used for the transformation to the output.

The exact form of the transformation applied to the final hidden state of the `[CLS]` token is a simple $d \times d$ linear

[1] Although the results are not directly comparable since Houlsby et al. (2019) use BERT-large and we use BERT-base.

[2] Vaswani et al. (2017) provide a more detailed motivation and discussion.

transformation, known as a **'pooling layer'**, followed by a nonlinearity then another matrix multiply that projects to the output space. The output space is always three dimensional or less in our case, and so this projection does not require many parameters. However separate pooling layers add d^2 parameters for each task. When sharing this layer we needed to use a non-standard training schedule; see section 4.1.

3.2. Adding Parameters to the Top

The simplest way to add parameters to a model is to add them at the 'top' of the model, i.e. just before the classification layer.

We get our final hidden state for `[CLS]`, h^f, from the original vector embeddings of the tokens in the sequence (of length l), $\{\mathrm{h}_t\}_{t=0}^{l}$, by

$$\mathrm{h}^f = \mathrm{TS}(\mathrm{BERT}(\{\mathrm{h}_t\}_{t=0}^{l})), \tag{6}$$

where $\mathrm{TS}(\cdot)$ is a task-specific function that can potentially operate on a single vector, but depends on the entire sequence when it contains attention layers. $\mathrm{BERT}(\cdot)$ always depends on the entire sequence, and is shared across tasks.

The benefits of this form are that at inference time we only apply $\mathrm{BERT}(\{\mathrm{h}_t\}_{t=0}^{l})$ once (assuming the setting where we perform multiple tasks on the same piece of text), which saves significantly on total operations because each $\mathrm{TS}(\cdot)$ requires much fewer operations than the main BERT model.

The simplest form for the task-specific transformation of the hidden state $\mathrm{TS}(\cdot)$ would be a linear transform followed by a nonlinearity. However this requires d_m^2 parameters, and d_m is fairly large even for BERT-base. The linear transform does not violate our 15 million parameter constraint, but we expect there are more efficient ways to add parameters.

Another obvious transformation, adding an extra BERT layer for each task, results in approximately a $1.67\times$ increase in number of parameters, or 73 million new parameters. d_{ff} is $4d_m$ for BERT, so for a BERT layer we get $4d_m^2 + 2d_m d_{ff} = 12d_m^2$ parameters. We include this architecture in our experiments for comparison, with the caveat that it requires many more parameters than our alternatives.

To avoid transformations requiring $O(d_m^2)$ parameters, we propose using task-specific functions of the form

$$\mathrm{TS}(\mathrm{h}) = V^D g(V^E \mathrm{h}), \tag{7}$$

where V^E is a $d_s \times d_m$ 'encoder' matrix, V^D is a $d_m \times d_s$ 'decoder' matrix with $d_s < d_m$, and $g(\cdot)$ is an arbitrary function. Because we can make d_s as small as we like, $g(\cdot)$ can be composed of multiple layers of transformations, and not impose a large parameter budget.

We experiment with these choices for each layer of $g(\cdot)$:

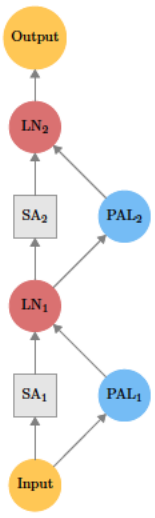


Figure 1. Schematic diagram of adding a task-specific function (here our 'Projected Attention Layers' or PALs) in parallel with self-attention (SA) layers in a BERT model (see section 3.3), with only two layers for simplicity. LN refers to layer-norm.

- Multi-head attention, optionally followed by a residual connection and layer-norm. We refer to this method as **Projected Attention**. We found $d_s = 204$ worked well, and allowed us to stay within our $1.13\times$ parameter limit.
- A one or two layer feed-forward network followed by a residual connection and layer-norm, such that it has the same number of parameters as the previous form; this means the intermediate layer is of size 408 (for a one layer network) or 252 (for a two layer network).

3.3. Adding Parameters within BERT

Instead of adding parameters to the top of the model, we may want to modify the $\mathrm{BERT}(\cdot)$ function itself, inspired by 'residual adapter modules' (section 2.1, Rebuffi et al., 2018). Specifically, we wish to add task-specific parameters to each layer of the BERT model. See figure 1 for an illustration.

We can add a task-specific function 'in parallel' with each BERT layer as follows:

$$\mathrm{h}^{l+1} = \mathrm{LN}(\mathrm{h}^l + \mathrm{SA}(\mathrm{h}^l) + \mathrm{TS}(\mathrm{h}^l)) \tag{8}$$

where l indexes the layer. This means we recover the original BERT model if $\mathrm{TS}(\cdot)$ outputs a zero vector. Alternatively we can add a 'serial' connection where we transform the output of a BERT layer:

$$\hat{\mathrm{h}}^{l+1} = \mathrm{LN}(\mathrm{h}^l + \mathrm{SA}(\mathrm{h}^l)) \tag{9}$$

$$\mathrm{h}^{l+1} = \mathrm{LN}(\hat{\mathrm{h}}^{l+1} + \mathrm{TS}(\hat{\mathrm{h}}^{l+1})). \tag{10}$$

In preliminary experiments, serial connections gave consistently much worse results than parallel connections, and we report results for parallel connections in what follows.

We again consider task-specific functions of the form:

$$\mathrm{TS}(\mathbf{h}) = V^D g(V^E \mathbf{h}), \tag{11}$$

with the difference that V^E (again a $d_s \times d_m$ matrix with $d_s < d_m$) and V^D (again a $d_m \times d_s$ matrix) are needed at each layer rather than only once each.

We experiment with $g(\cdot)$ taking the following forms:

- The identity function; This means our task-specific transform is just a low-rank linear transformation at each layer. To satisfy our parameter constraint we need $d_s = 100$. We refer to this method as **Low-rank Layers**.
- Multi-head attention. To satisfy our parameter constraint we need $d_s = 84$. We found that it was not necessary to use the W^o matrix (see section 3.1) when adapting within BERT, and did not use it in any of our models.
- Multi-head attention, with shared V^E and V^D across layers (not tasks). This parameter sharing allows a larger $d_s = 204$. We refer to this method as **Projected Attention Layers (PALs)**.
- Shared V^E and V^D across layers, but with $g(\cdot)$ a feed-forward network with intermediate size 306 instead of attention (and again $d_s = 204$).

The motivation behind PALs is that we want to spend our parameter budget on transformations with an inductive bias useful for sequences. The 'encoder' and 'decoder' matrices operate on each sequence element separately, unlike attention, which transforms the input based on the entire sequence. Finally, the attention mechanism of PALs can potentially be inspected to see which tokens in a sequence the task-specific parts of the model focus on, although we did not concentrate on this aspect in this work.

4. Multi-task Training and Experiment Setup

4.1. Sampling Tasks

A simple way to train a model on several tasks is to select a batch of training examples from each task, cycling through them in a fixed order. We refer to this as 'round-robin' sampling. However if the tasks have different numbers of training examples, round-robin sampling may not work well. By the time we have seen every example from a particular task we could have looped through another task's smaller dataset many times. This imbalance could lead to overfitting on smaller tasks, and under-training on larger tasks. Potentially we could alleviate this issue by manually tuning regularisation hyper-parameters for each task.

Table 1. How parameters are 'spent' for some of our methods, where T is the number of tasks, and there are 12 layers in the base network. The $2d_m d_s$ terms come from 'encoder' and 'decoder' matrices. PALs (section 3.3) use $3d_s^2$ parameters per multi-head layer (see section 3.1) rather than $4d_s^2$ because they do not use the final linear transform W^o. Projected attention (section 3.2) worked best with six rather than twelve layers.

METHOD	PARAMETERS
PALS	$T(2d_m d_s + 12 \times 3d_s^2)$
LOW RANK	$T(12 \times 2d_m d_s)$
PROJ. ATTN. ON TOP	$T(2d_m d_s + 6 \times 4d_s^2)$

Alternatively we can use methods where we see more examples from tasks with larger associated datasets. Concretely, we select a batch of examples from task i with probability p_i at each training step, and set p_i proportional to N_i, the number of training examples for task i:

$$p_i \propto N_i. \tag{12}$$

This is the approach of the multi-task BiLSTM of Wang et al. (2018a) on the GLUE benchmark, and was used by Sanh et al. (2018). It has the appealing property of selecting each example with the same probability as combining all the tasks and picking examples uniformly (though we train on batches from each task not single examples).

Since the ratio of the largest to the smallest task sizes N_i we use is ≈158, we only rarely train on some tasks with the simple $\propto N_i$ method. Training on one task (or a particular subset of tasks) for many steps can lead to interference, where performance on the other tasks suffers. A more general approach to sampling tasks sets p_i as:

$$p_i \propto N_i^\alpha. \tag{13}$$

If we choose $\alpha < 1$ we reduce the disparity between the probabilities of choosing tasks. We consider $\alpha = 0.5$ in our experiments, and call this method **'square root sampling'**.

Finally, we noticed that it was beneficial to train on tasks more equally towards the end of training, where we are most concerned about interference, and so we constructed the **'annealed sampling'** method where α changes with each epoch e:

$$\alpha = 1 - 0.8\frac{e-1}{E-1}, \tag{14}$$

where E is the total number of epochs. Since we used multiple datasets we chose a somewhat arbitrary 'epoch' of 2400 training steps.

It was particularly important to use the square root or annealed sampling methods when sharing a pooling layer (see section 3.1), and it makes intuitive sense that when the layer just before the output is shared, we need to guard against interference between tasks.

4.2. Setup

We based our experiments on the PyTorch implementation of BERT [3] and open-source our code[4]. No matter how we sampled tasks, we (unless stated otherwise) trained for 60,000 steps, with a minibatch size of 32, and a maximum sequence length of 128 tokens, choosing the best model from within that training time based on average development set score. We use Adam with learning rate of 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, L2 weight decay of 0.01, learning rate warmup over the first 10% of steps (usually 6,000), and linear decay of the learning rate after this, going down to zero at the end of training. We note warmup followed by linear decay is the 'slanted triangular learning rate' of Howard & Ruder (2018), who find it is suited for fine-tuning a language model on single tasks. We performed most of our experiments using either the 'proportional', 'square root' or 'annealed' sampling methods (see section 4.1). Round robin sampling gave consistently worse results.

We use twelve heads for the attention mechanism in PALs and other methods, except when using a smaller hidden size, where we decreased it proportionally. We did not find significant performance differences when changing the number of heads. We used the same BERT-base architecture as by Devlin et al. (2018), twelve attention heads, $d_{ff} = 3072$ and $d_m = 768$ (see section 3.1).

We found it was crucial to use the pre-trained weights for BERT-base and not start from scratch. When training from scratch, with adaption parameters or not, we got significantly worse performance. For some tasks we did not get better results than random guessing after 90,000 steps. Although we note we used the same hyper-parameters as when training from the pre-trained weights, which might not be optimal for starting from scratch. We experimented briefly with freezing the BERT-base parameters and fine-tuning only the PALs and alternatives, but concentrated on training all of the parameters, finding it took less parameters to approach matching fine-tuned BERT.

4.3. Details of GLUE Tasks

We test our methods for multi-task adaptation on eight of the nine tasks in the GLUE benchmark (Wang et al., 2018a)[5].

[3]https://github.com/huggingface/pytorch-pretrained-BERT

[4]https://github.com/AsaCooperStickland/Bert-n-Pals

[5]Wang et al. (2018a) provide a more detailed discussion of these tasks.

Single-sentence tasks: Acceptability classification with CoLA (Warstadt et al., 2018); binary sentiment classification with SST (Socher et al., 2013).

Sentence pair tasks: Semantic similarity with the MSR Paraphrase Corpus (MRPC: Dolan & Brockett, 2005), STS-Benchmark (STS: Cer et al., 2017) and Quora Question Pairs (QQP) dataset, and textual entailment with Multi-Genre NLI Corpus (MNLI: Williams et al., 2018), a subset of the RTE challenge corpora (Dagan et al., 2006), and data from SQuAD (QNLI: Rajpurkar et al., 2016).

Like Devlin et al. (2018) we exclude the Winograd NLI task. When systems are trained on this task they have always performed worse than the 65.1 baseline accuracy of predicting the majority class. For our submissions we also simply predicted the majority class.

5. Experiments and Discussion

Table 2 lists our results on GLUE for our best-performing PAL model (chosen by average development set performance), and some alternatives. Our main comparison is against fine-tuned BERT-base, which in the absence of transfer effects represents an upper bound on our performance, since it involves tuning all BERT-base parameters to perform well on each task individually, therefore requiring approximately $8\times$ as many parameters as our methods. By construction, apart from our adaptation parameters we use the exact same architecture as BERT-base. We note that with the exception of our results for RTE, better performance can be obtained by fine-tuning the BERT-large model that has approximately $3\times$ the parameters of BERT-base.

The use of multi-task training significantly improves results on the RTE task, achieving state-of-the-art performance. Similar improvements have been observed with multi-task LSTM-based systems (Wang et al., 2018a) and by pre-training on MNLI before fine-tuning on RTE (Phang et al., 2018). Since RTE has the smallest number of training examples, and is similar to MNLI, it makes intuitive sense that it benefits from multi-task training. Sharing more parameters increased performance on RTE, and our fully-shared model has slightly better performance on RTE than PALs, however PALs are the only model that matches BERT-base on the larger tasks as well as performing well on RTE.

For the large sentence-pair tasks, MNLI, QQP and QNLI, performance is almost exactly the same as BERT-base with PALs. For the two single sentence tasks: the syntax-oriented CoLA task and the SST sentiment task we see the largest drops in performance with PALs. This is in agreement with the results of Phang et al. (2018) who did not observe any transfer from various intermediate tasks, and, for CoLA, mirrors the results of Bowman et al. (2018) that language modeling alone is the best pre-training task for CoLA.

Table 2. GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. We show F1/accuracy scores for QQP and MRPC, and accuracy on the matched/mismatched test sets for MNLI. The 'Av.' column is slightly different than the official GLUE score, since we exclude WNLI. 'Bert-base' results are from Devlin et al. (2018). 'Shared' refers to the model where all parameters are shared except the final projection to output space. The models we tested are a result of the 'annealed sampling' method for multi-task training as it produced the best results on the dev set.

METHOD	PARAMS	MNLI-(M/MM) 392K	QQP 363K	QNLI 108K	SST-2 67K	COLA 8.5K	STS-B 5.7K	MRPC 3.5K	RTE 2.5K	AV.
BERT-BASE	8×	84.6/83.4	89.2/71.2	90.1	93.5	52.1	85.8	84.8/88.9	66.4	79.6
SHARED	1.00×	84.0/83.4	88.9/70.8	89.3	93.4	51.2	83.6	81.3/86.7	76.6	79.9
TOP PROJ. ATTN.	1.10×	84.0/83.2	88.8/71.2	89.7	93.2	47.1	85.3	83.1/87.5	75.5	79.6
PALS (204)	1.13×	84.3/83.5	89.2/71.5	90.0	92.6	51.2	85.8	84.6/88.7	76.0	**80.4**

Table 3. GLUE performance, in terms of average score across each task's development set; this score is accuracy except for CoLA, where it is Matthews correlation, and STS-B, where it is Pearson correlation. We show the mean and standard error over three random seeds, unless standard error is < 0.005. For the details of the sampling strategies see section 4.1. For the 'within BERT' methods we show the smaller hidden state size in brackets, and write 'no sharing' to refer to not sharing V^E and V^D across layers, 'top' to mean adding in parallel to the six BERT layers just before the output, and 'bottom' to mean adding in parallel to the six BERT layers just after the input.

METHOD	NO. PARAMS	NEW LAYERS	PROP. SAMP.	SQRT. SAMP.	ANNEAL SAMP.
SHARED	1.00×	0	79.17±0.03	80.56±0.04	80.7±0.3
ADDING ON TOP OF BERT					
BERT LAYER	1.66×	1	80.6±0.2	81.6±0.3	81.5±0.2
PROJ. ATTN.	1.10×	6	80.3±0.1	81.4±0.1	81.5±0.1
PROJ. FFN (1 LAYER)	1.10×	6		81.07	80.8±0.1
ADDING WITHIN BERT					
PALS (204)	1.13×	12	80.6±0.2	81.0±0.2	**81.7±0.2**
PALS NO SHARING (84)	1.13×	12			81.3±0.1
LOW RANK (100)	1.13×	12			**81.9±0.2**
PALS (276, TOP)	1.13×	6			81.61±0.06
PALS (276, BOTTOM)	1.13×	6			81.4±0.1

5.1. PALs and Alternatives

Table 4 lists our results on the GLUE benchmark development set for various ways of adding task-specific parameters and sampling strategies.

Our best results came with PALs, or low-rank layers, adapting every layer within BERT. The performance of PALs increased with a larger hidden state. Having separate 'encoder' and 'decoder' matrices (see section 3.3) across layers, or having separate pooling layers for each task, with the appropriate reduction in hidden state size to make up for the extra parameters, resulted in worse performance for PALs. However sharing 'encoder' and 'decoder' matrices between *tasks* or both layers and tasks hurt results. A larger hidden state size seems important for Transformer models, e.g. the performance of BERT-large vs. BERT-base (Devlin et al., 2018) or the ablation study by Vaswani et al. (2017).

We tested two adaption layers that did not use attention: Low-rank layers, and our method with shared 'encoder' and 'decoder' matrices but with a small feedforward network in-between them instead of attention. The latter model did not achieve good performance, but low-rank layers and PALs have similar mean performance.

By inspecting the best-performing single models of each method we see a contrast: the strong results for low-rank layers are partly from better performance on CoLA. CoLA tends to see larger changes in score between models than other tasks since it is scored by a different measure (Matthews correlation coefficient rather than accuracy). PALs performed better for the three largest tasks, MNLI, QQP and QNLI, and equivalently for other tasks.

These results suggest PALs has greater representational capacity; the only model that achieved comparable performance on the large tasks was adding an entire BERT-layer to the top, but this model had worse performance on the RTE task and uses many more parameters. The fact that spending parameters on linear transforms in the encoder, decoder or pooling matrices gives worse performance, and the

worse performance of feedforward layers compared to multi-head attention, points towards the inductive bias provided by attention being important for good performance.

However at sufficiently parameter constrained regimes (for example 1.5 million parameters, which implies $d_s = 10$ for low-rank transforms, and $d_s = 60$ for PALs), PALs and low-rank layers performed similarly to the fully-shared model. Using the LHUC method (see section 2.1), which requires even fewer parameters, also gave no improvement over the fully-shared baseline.

Ultimately, given the simplicity and competitive performance of low-rank layers, they remain an attractive option. There may be bigger differences for tasks like question answering which rely on the hidden states of every token in the input (as opposed to GLUE tasks which only use the final `[CLS]` hidden state to make predictions). We note that PALs and low-rank layers can easily be combined, say by using one type of adapter in the higher layers of the network and another in the lower ones.

When adding parameters to the top of BERT-base, it was important to use attention rather than feedforward transforms. Six additional layers worked best, outperforming using twelve or three layers. We also found it was crucial to use layer-norm and residual connections after each application of attention. Surprisingly, for these models using a separate pooling layer did not noticeably change results, and we report results with a shared pooling layer, which requires fewer parameters. These models saw worse performance on the RTE task, perhaps because transfer from other tasks is important, and splitting the model into multiple 'heads' for each task dampens the benefits of shared knowledge.

5.2. Where should we add Adaptation Modules?

We draw some of the same conclusions as Rebuffi et al. (2018) for 'residual adapter modules'. As that work studied multi-task computer vision with residual networks (section 2.1), we hope that these principles will apply broadly.

Adding task-specific functions *within* networks works better than adding them to the *top* (for a given number of parameters). As found by Rebuffi et al. (2018), the best performing models had adaptations at *every layer* of the base network, and adding adapter modules to the *final half* of the base model worked better than adding to the half *just after the input*. Unfortunately, adapting every layer of the base model represents the worst case for sharing operations between tasks. (We note again that this sharing is possible only when we want to perform many tasks on the same piece of text). But adapting the final half achieved slightly better performance than adding to the top of BERT-base. When adapting the final half we can still share the first six layers worth of operations, offering a useful compromise.

For within-network adaptations, *parallel* connections worked better than *serial* ones, also as found by Rebuffi et al. (2018). Our results with serial connections were much worse than simply not including any adapters. While the parallel configuration acts as a perturbation on the base network, the serial configuration more directly changes the hidden states being fed into the next layer. In these ways, the parallel configuration is less prone to the loss of the 'knowledge' stored in the base network. We note that our serial configuration adds a newly initialised layer-norm, which may be the source of the performance drop.

6. Further Discussion

We found the details of how to schedule training examples from each task were important. With a lot of parameter sharing, sampling tasks proportional to dataset size impaired performance compared to our 'annealing' method, where we slowly decrease the influence of dataset size on sampling probability. Annealing increased the variance of performance across random seeds as well as mean performance, meaning that we may need to pay the cost of several training runs to obtain the best single models from this method. We did not consider many variations of training method, and used no methods to reduce interference from training on separate tasks (to take one example, the 'Gradient Episodic Memory' of Lopez-Paz & Ranzato, 2017). How these methods interact with choice of adaptation parameters is a direction for further research.

We introduced 'Projected Attention Layers' as a transformation that can adapt the BERT sentence representation model for multi-task learning. PALs give a higher capacity for a given number of parameters compared to all the alternatives we considered, although simple low-rank transformations remain attractive due to their simplicity. If we adapt all the layers of BERT-base, we cannot share any operations across tasks. Ultimately the choice of which method to use depends on the constraints in place; if parameters are less constrained but you want to share as many operations as possible, adding an entire task-specific BERT layer on top of the model makes sense. If shared operations are not an issue, then adding PALs to every layer will perform well with few parameters. Finally, adapting only the final half of the base model offers a compromise between performance and sharing operations.

Acknowledgements

We would like to thank Ivan Titov and Timothy Hospedales for useful discussion, and Elaine Farrow for help with a draft version of this paper. Asa Cooper Stickland was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical

Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

References

Ba, J., Kiros, R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Bowman, S. R., Pavlick, E., Grave, E., Durme, B. V., Wang, A., Hula, J., Xia, P., Pappagari, R., McCoy, R. T., Patel, R., Kim, N., Tenney, I., Huang, Y., Yu, K., Jin, S., and Chen, B. Looking for ELMo's friends: Sentence-level pretraining beyond language modeling. *CoRR*, abs/1812.10860, 2018.

Caruana, R. Multitask learning. *Mach. Learn.*, 28(1): 41–75, July 1997. ISSN 0885-6125. doi: 10.1023/A:1007379606734.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pp. 1–14. Association for Computational Linguistics, 2017. doi: 10.18653/v1/S17-2001.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12: 2493–2537, November 2011. ISSN 1532-4435.

Dagan, I., Glickman, O., and Magnini, B. The pascal recognising textual entailment challenge. In *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, MLCW'05, pp. 177–190, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-33427-0, 978-3-540-33427-9. doi: 10.1007/11736790_9.

Dai, A. M. and Le, Q. V. Semi-supervised sequence learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 3079–3087. Curran Associates, Inc., 2015.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. Transformer-XL: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

Deng, L., Hinton, G., and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: an overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603, May 2013. doi: 10.1109/ICASSP.2013.6639344.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Duong, L., Cohn, T., Bird, S., and Cook, P. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 845–850. Association for Computational Linguistics, 2015. doi: 10.3115/v1/P15-2139.

Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1923–1933. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1206.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *ECCV*, 2016.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-Efficient Transfer Learning for NLP. *CoRR*, abs/1902.00751, 2019.

Howard, J. and Ruder, S. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 328–339. Association for Computational Linguistics, 2018.

Liu, X., He, P., Chen, W., and Gao, J. Multi-task deep neural networks for natural language understanding. *CoRR*, abs/1901.11504, 2019.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6467–6476. Curran Associates, Inc., 2017.

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

Phang, J., Févry, T., and Bowman, S. R. Sentence encoders on STILTSs: Supplementary training on intermediate labeled-data tasks. *CoRR*, abs/1811.01088, 2018.

Radford, A. Improving language understanding by generative pre-training. 2018.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1264.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. Efficient parametrization of multi-domain deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2 2018.

Ruder, S. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

Sanh, V., Wolf, T., and Ruder, S. A hierarchical multi-task approach for learning embeddings from semantic tasks. *CoRR*, abs/1811.06031, 2018.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. Association for Computational Linguistics, 2013.

Subramanian, S., Trischler, A., Bengio, Y., and Pal, C. J. Learning general purpose distributed sentence representations via large scale multi-task learning. In *International Conference on Learning Representations*, 2018.

Swietojanski, P. and Renals, S. Learning hidden unit contributions for unsupervised speaker adaptation of neural network acoustic models. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pp. 171–176, Dec 2014. doi: 10.1109/SLT.2014.7078569.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multitask reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4496–4506. Curran Associates, Inc., 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355. Association for Computational Linguistics, 2018a.

Wang, X., Girshick, R., Gupta, A., and He, K. Non-local neural networks. *CVPR*, 2018b.

Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *CoRR*, abs/1805.12471, 2018.

Williams, A., Nangia, N., and Bowman, S. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-1101.

Yang, Y. and Hospedales, T. M. Trace norm regularised deep multi-task learning. In *ICLR Workshop*, 2017.

Zellers, R., Bisk, Y., Schwartz, R., and Choi, Y. Swag: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Zhang, H., Goodfellow, I. J., Metaxas, D. N., and Odena, A. Self-attention generative adversarial networks. *CoRR*, abs/1805.08318, 2018.

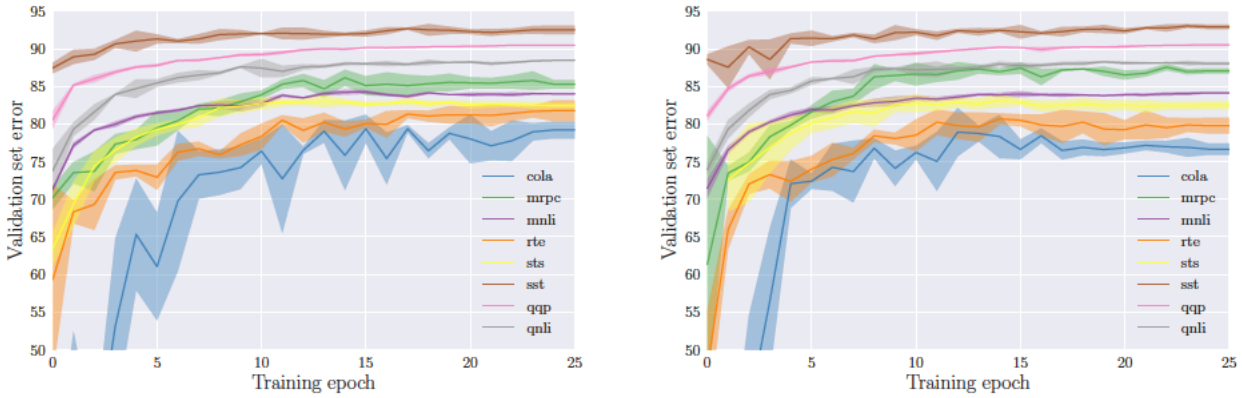
A. Performance on Tasks Over Time

Figure 2 shows performance on the GLUE tasks over time for PALs and low-rank adapter modules. The low-resource tasks have a much larger variation in performance than the high resource ones, which are fairly stable. CoLA performance in particular varies a lot early on in training. Performance on CoLA and RTE goes down towards the end of training with low-rank adapters, and not with PALs, and the opposite trend for MRPC. These downward trends might be rectified with a better training schedule or regularisation scheme.

B. Squad and SWAG Performance

We conducted limited experiments on two additional tasks. The Stanford Question Answering Dataset (SQuAD) is a collection of 100k crowdsourced question/answer pairs (Rajpurkar et al., 2016), where the task is to predict the location of the answer in a paragraph from Wikipedia. We follow the approach of Devlin et al. (2018) by associating each token in the input sequence with a probability of being the start, and end, of the answer span. The Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples intended to evaluate grounded commonsense inference (Zellers et al., 2018). Given a sentence from a video captioning dataset, the task is to decide among four choices the most plausible continuation, with each sentence-completion pair assigned a score, and a softmax applied over the four choices to form a probability distribution.

We tested multi-task learning with the SQuAD and SWAG datasets. We follow all the same experimental settings as before, but we use round robin sampling because of the comparable size of the datasets, and train for 24,000 steps, not 60,000, with an increased maximum sequence length, 256. Results, see table 4, show a slight improvement when using the PAL adapters compared to a fully shared baseline and low-rank adapters. However all approaches performed similarly, with there perhaps less need for the flexibility provided by adapters when only training on two tasks.



(a) PALs (b) Low rank adapters

Figure 2. Average performance over four random seeds for two adapter modules, with the shaded region indicating standard deviation. CoLA performance has been shifted up by 30% for visibility.

Table 4. Performance on SQuAD and SWAG, in terms of average score across each task's development set; this score is exact match and f1 score for SQuAD, and accuracy for SWAG.

METHOD	NO. PARAMS	NEW LAYERS	ROUND ROBIN
SHARED	1.00×	0	82.75±0.09
ADDING WITHIN BERT			
PALS (204)	1.13×	12	82.774±0.006
LOW RANK (100)	1.13×	12	82.74±0.06

3.2 Contribution and impact

This paper had 173 citations on 2023-04-04 according to Google Scholar. The methods introduced by the paper were used as a baseline for subsequent work with a per-task parameters for multi-task learning (Pfeiffer et al., 2021; Tay et al., 2020). The idea of per-task parameters was extended to per-language parameters for multilingual NLP (Pfeiffer et al., 2020; Philip et al., 2020) and per-language + per-domain parameters for multilingual, multi-domain machine translation (Stickland et al., 2021a). Our 'low-rank' layers are very similar to the later LoRA (Hu et al., 2021) method (see also the next chapter).

As noted before, concurrent work (Houlsby et al., 2019) explored similar ideas to our paper. The main differences are as follows:

- Houlsby et al. (2019) focus on fine-tuning BERT on a single task while only updating a small number of parameters, whereas we focus on multi-task learning, reducing interference with task-specific parameters.
- Relatedly, Houlsby et al. (2019) freeze (i.e. don't calculate gradients for or update) the backbone BERT model, only training 'adapters', which are small modules inserted after each layer similar to our 'low-rank layers'. This is required for maintaining a low parameter storage cost, since the (large) frozen backbone is shared across all tasks. We do not freeze the backbone since it is shared across all tasks during multi-task learning and therefore we can still maintain a low parameter storage cost even while updating every parameter.
- Our 'low-rank' layers update the final hidden states $\vec{h}$ of a layer (before layer-norm and a residual connection) as $\vec{h} \leftarrow \vec{h} + V^D V^E \vec{x}$ where $\vec{x}$ is the input to the layer, whereas Houlsby et al. (2019) use $\vec{h} \leftarrow \vec{h} + V^D \text{ReLU}(V^E \vec{h})$, i.e. a feed-forward network with a small hidden state, and $\vec{h}$ is the input to the 'adapter', not $\vec{x}$.
- Houlsby et al. (2019) add adapters at every residual connection, i.e. after both the self-attention and feed-forward modules of every layer, whereas we add parameters once per layer, after the feed-forward module.

Freezing the backbone model ensures no information is lost while fine-tuning and reduces training cost by avoiding calculating gradients for most model parameters, however it is impossible to get positive transfer between tasks with the method of Houlsby et al. (2019) since 'adapters' are trained on each task separately. Several works combine multi-task learning and freezing most model parameters, by using 'hypernetworks'. In this setup, adapter parameters are generated as a function (the 'hypernetwork') of task or language embeddings (i.e. each task has an associated learnable vector). Examples include Karimi Mahabadi et al. (2021a); Tay et al. (2020); Üstün et al. (2020).

Since the publication of this work, the scale of pre-trained models has increased from ≈ 300 million to many billions of parameters, and new parameter efficient methods have been proposed. In the next chapter we explore connections between these methods and evaluate them empirically on modern, large-scale pre-trained models. We concentrate on the 'frozen backbone' paradigm of Houlsby et al. (2019) since it is 1) simple, avoiding design choices such as choosing tasks which may have positive transfer and scheduling tasks to avoid overfitting to less data-rich tasks and 2) more popular in practice than the multi-task learning setup. This simplicity additionally allows us to test more parameter-efficient method architectures.

Chapter 4

Survey of Parameter-efficient Fine-tuning Methods

This chapter is dedicated to a survey of parameter-efficient fine-tuning methods (hereafter referred to as 'PEFT methods'), including various alternatives to the architectures introduced in the previous chapter. We empirically test the performance of these architectures on English text classification with the popular 'T5' (Raffel et al., 2019) model family as the base model, and focus on the 'frozen backbone' paradigm due to its popularity and simplicity. We show empirically (§ 4.3.1) how PEFT methods save on memory at fine-tuning time, and can sometimes outperform full fine-tuning. We make connections between methods, and show how they are related to methods proposed outside the context of NLP and transformers (§ 4.1.2 and § 4.1.4).

A large part of this thesis is dedicated to a particular PEFT method, so-called 'adapters'. A version of adapters without a non-linearity, so called 'low-rank layers' were introduced in the previous chapter. The 'Low-Rank Adaptation' or LoRA method (Hu et al., 2021) is architecturally almost equivalent to 'low-rank layers' but Hu et al. (2021) makes some useful additions, discussed in the next section. We validate that 'adapters', and the LoRA variant perform well under the conditions of modern, large-scale models, with empirical results on PLMs ranging from 60 million to 3 billion parameters. We provide a guide of when to use adapters vs. other methods.

A number of other surveys of (Ding et al., 2022), comparisons of (He et al., 2022; Karimi Mahabadi et al., 2021b) and tutorials on (Ruder et al., 2022) PEFT methods exist. We also recommend watching the video version of Ruder et al. (2022): `https://www.youtube.com/watch?v=KoOlcX3XLd4`. We aim to distinguish this survey by a focus on methods that can be described by a 'perturbation' on top of base model representations (i.e. model hidden states) $\vec{h}$: $\vec{h} \leftarrow \vec{h} + f(\vec{h})$ for some function $f(\cdot)$. This is because this thesis largely deals with these methods, they have been widely used by the open source community recently (for example in the PEFT library [1]), and they are simple to describe and compare. We also focus on testing the 'scaling' behavior of these methods, i.e. how does their performance change as we increase the size of the base pre-trained model.

4.1 Connections Between PEFT Methods

4.1.1 Existing Parameter-efficient Tuning Methods

Below we review several parameter-efficient tuning methods. Unless otherwise specified, all pre-trained model parameters are frozen and only parameters introduced by a given method are

[1] `https://github.com/huggingface/peft`

tuned.

Adapters (Houlsby et al., 2019): We briefly described this approach in the last chapter and now introduce it in more detail. The adapter approach inserts small modules (adapters) between transformer layers. The adapter layer generally uses a down-projection with $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d \times r}$ to project the input $\vec{h}$ to a lower-dimensional space specified by bottleneck dimension r, followed by a nonlinear activation function $f(\cdot)$, and a up-projection with $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r \times d}$. These adapters are surrounded by a residual connection, leading to a final form:

$$\vec{h} \leftarrow \vec{h} + f(\vec{h}\mathbf{W}_{\text{down}})\mathbf{W}_{\text{up}}. \tag{4.1}$$

Houlsby et al. (2019) places two adapters sequentially within one layer of the transformer, one after the multi-head attention and one after the FFN sub-layer. There are many variations on this basic idea, for example Pfeiffer et al. (2021) propose an efficient adapter variant that is inserted only after the FFN "add & layer norm" sub-layer (see Section 2.3.1 for a description of the transformer layer).

This chapter focuses on a particularly simple implementation with one adapter per transformer layer, where the input to the adapter $\vec{h}$ is the output of the feed forward network sub-layer, and we simply add the adapter output to the final output of the layer. A visual representation is given in Figure 4.2. In some transformer variants we would apply normalization at the end of the layer (post-adapter), but the T5 models we use in this chapter apply normalization before each sub-layer.

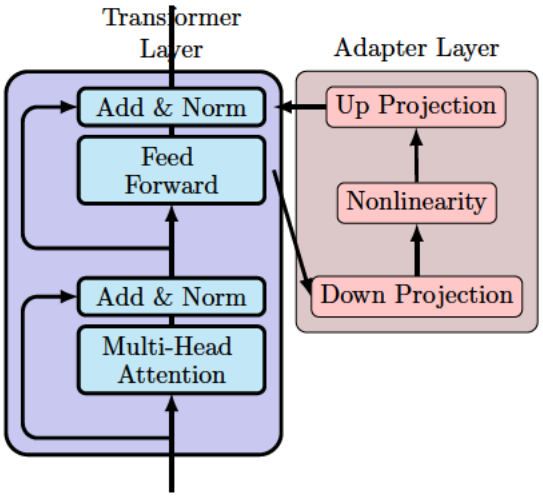


Figure 4.1: Schematic diagram of an adapter layer for a transformer where the normalization layer comes after the sub-layer. The feed forward network output is fed into the adapter, then the sum of a residual connection to the feed forward network input, the original feed forward network output and finally the adapter output are all fed into the normalization layer.

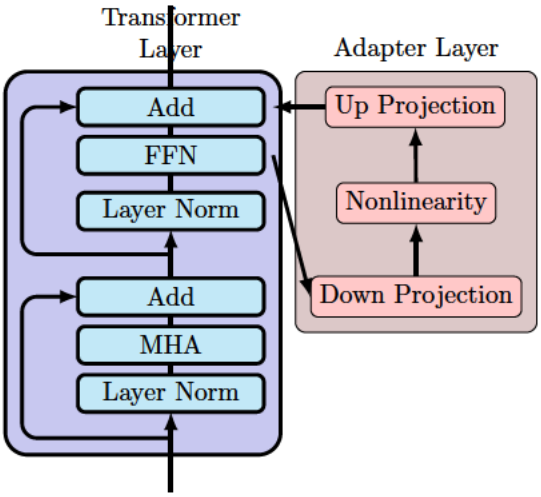


Figure 4.2: Schematic diagram of an adapter layer for a transformer where the normalization layer comes before the sub-layer, like the T5 family models we consider in this chapter. The feed forward network output is fed into the adapter, then the sum of a residual connection to the feed forward network input, the original feed forward network output and finally the adapter output are all fed into the next layer.

Prefix Tuning (Li and Liang, 2021): Inspired by the success of textual prompting methods (Liu et al., 2021a), prefix tuning prepends p tunable prefix vectors to the keys and values of the

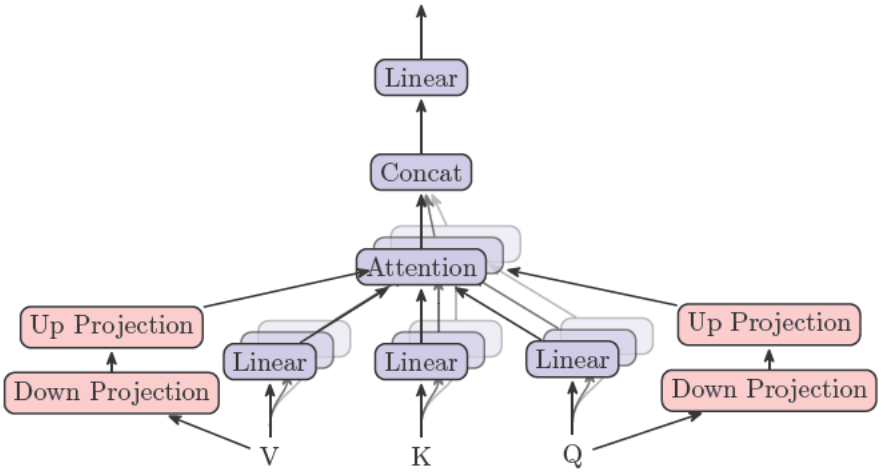


Figure 4.3: Schematic diagram of the LoRA method when applied to the value (V) and query (Q) matrices. In addition to the usual linear transform before the attention mechanism, we add a low rank transformation. We leaving the new transformation trainable, and freezing all other parameters.

multi-head attention at every layer. Specifically, two sets of prefix vectors $\vec{p}^k, \vec{p}^v \in \mathbb{R}^{p \times d}$ are split into N_h pieces, $\vec{p}^{k(e)}, \vec{p}^{v(e)} \in \mathbb{R}^{p \times d/N_h}$, one for each head, and concatenated with the original keys and values. Then multi-head attention is performed on the new keys and values which each have p extra 'virtual' token embeddings. The computation of head e in Eq. 2.5 becomes:

$$\vec{h}_j^e = \sum_{i=0}^{L} \text{softmax}((\mathbf{W}_k^e \vec{h}_i)^T (\mathbf{W}_q^e \vec{h}_j)) \mathbf{W}_v^e \vec{h}_i + \sum_{i=L}^{L+p} \text{softmax}((\vec{p}_j^{k(e)})^T (\mathbf{W}_q^e \vec{h}_j)) \vec{p}_j^{v(e)}, \tag{4.2}$$

with the right-most sum representing the perturbation from prefix-tuning. Prompt-tuning (Lester et al., 2021) simplifies prefix-tuning by only prepending to the input word embeddings in the first layer; similar work also includes P-tuning (Liu et al., 2021b).

LoRA (Hu et al., 2021): LoRA injects trainable low-rank matrices into transformer layers to approximate the weight updates. For a pre-trained weight matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$, LoRA represents its update with a low-rank decomposition $\mathbf{W} + \Delta W = \mathbf{W} + \mathbf{W}_{\text{down}} \mathbf{W}_{\text{up}}$, where $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d \times r}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{r \times k}$ are tunable parameters. LoRA applies this update to the query and value projection matrices $(\mathbf{W}_q, \mathbf{W}_v)$ in the multi-head attention sub-layer. For a specific input $\vec{x}$ to the linear projection in multi-head attention, LoRA modifies the projection output $\vec{h}$ as:

$$\vec{h} \leftarrow \vec{h} + s \cdot \vec{x} \mathbf{W}_{\text{down}} \mathbf{W}_{\text{up}}, \tag{4.3}$$

where $s \geq 1$ is a tunable scalar hyperparameter, typically parameterized in terms of the rank of the low-rank decomposition, $s = g/r$ with g a tunable parameter and r rank[2]. For a visual representation see Figure 4.3. We can recover the base model architecture exactly by '**folding in**' the low-rank update $\mathbf{W} \leftarrow \mathbf{W} + s \cdot \vec{x} \mathbf{W}_{\text{down}} \mathbf{W}_{\text{up}}$. This means no extra computational at inference time; the same procedure is impossible for other methods like FFN adapters which introduce non-linearities.

We briefly contrast this architecture to the 'low-rank layers' introduced in the previous chapter. They correspond to removing the scaling constant s, so $\vec{h} \leftarrow \vec{h} + \cdot \vec{x} \mathbf{W}_{\text{down}} \mathbf{W}_{\text{up}}$, and setting $\vec{h}$ to

[2]The public code of LoRA at https://github.com/microsoft/LoRA uses different s in different datasets, and we have verified the value of s could have a significant effect on the results.

the final hidden state of the layer and $\vec{x}$ as the first hidden state of the layer, and so acting in parallel to the entire layer, not just one linear transform. This makes the 'folding in' property impossible due to the many nonlinearities in a transformer layer.

Batch Ensemble (Hu et al., 2021): Batch Ensemble was originally introduced in the context of ensembles of models. Independently training n models and averaging their predictions often generalises better and is better calibrated than a single model, but of course increases training and inference cost n times. Batch Ensemble is a more efficient way of producing an ensemble, sharing most parameters across all ensemble members but with low rank updates for each ensemble member; However there is no reason we cannot use the same idea for parameter-efficient fine-tuning. For a pre-trained weight matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$, we update with the Hadamard product with a rank one matrix decomposition $\mathbf{W} \circ \Delta W = \mathbf{W} \circ \vec{u}\vec{v}^T$, where $\vec{u} \in \mathbb{R}^d, \vec{v} \in \mathbb{R}^k$ are tunable parameters. Batch Ensemble can be vectorised in order to avoid explicitly calculating $\vec{u}\vec{v}^T$ For a specific input $\vec{x}$ to a linear projection, Batch Ensemble modifies the projection output $\vec{h}$ as:

$$\vec{h} \leftarrow (\vec{x} \circ \vec{u})\mathbf{W} \circ \vec{v}, \tag{4.4}$$

and we found it useful to parameterize this as

$$\vec{h} \leftarrow (\vec{x} \circ (\vec{1} + s_1 \cdot \vec{u}))\mathbf{W} \circ (\vec{1} + s_2 \cdot \vec{v}), \tag{4.5}$$

where $s_i \geq 1$ are tunable scalar hyperparameters and $\vec{1}$ is a vector with every entry set to one. While we focus on methods proposed in NLP, various methods have been proposed in other domains, and we can make connections to the above formulation. LHUC (Learning Hidden Unit Contributions) was introduced in the speech domain, with the idea of adapting to the style of a particular speaker (Swietojanski et al., 2016). LHUC updates the hidden states of a speech model by multiplication with a vector, like our $\vec{v}$, with the slight modification of also applying a sigmoid function (and multiplying by two, constraining the elements of $\vec{v}$ between 0 and 2).

4.1.2 Connection to 'adapters' in computer vision

The low-rank update $\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$ is similar to the 'parallel residual adapters' introduced by Rebuffi et al. (2018). Rebuffi et al. (2018) adapt a CNN trained on the more general ImageNet image classification dataset to a specific domain (e.g. pictures of flowers). Rather than update a standard linear layer, Rebuffi et al. (2018) updates the convolutional filter $\mathbf{f}$ of a standard ResNet architecture,

$$\mathbf{h} \leftarrow \mathbf{f} * \mathbf{x} + \mathbf{A}\mathbf{x} = \mathbf{h} + \mathbf{A}\mathbf{x}, \tag{4.6}$$

where $\mathbf{x}, \mathbf{h} \in \mathbb{R}^{L\times L\times d}$ are input and output hidden states with height and width L and dimension d, for a particular layer. Here the update $\mathbf{A} \in \mathbb{R}^{d\times d}$ is written as a matrix multiplication of the hidden dimension d, but can alternatively be thought of as a 1×1 convolution operation, and can thus be '**folded in**' to the standard convolution with $\mathbf{f}$. Rebuffi et al. (2018) propose reducing the parameter count of $\mathbf{A}$ by taking $\mathbf{A}$ after it is trained on the domain adaptation task and use a singular value decomposition (SVD) to replace $\mathbf{A}$ by a low-rank approximation $\mathbf{A} \approx \mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$. The $\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$ matrices are then further fine-tuned on the domain adaptation task. Without the fine-tune + SVD step to initialise the low-rank approximation, this is exactly the same as the LoRA method, except applied to a convolutional filter instead of a full rank matrix.

Others: Other parameter-efficient tuning methods include BitFit (Ben Zaken et al., 2022), which only fine-tunes bias vectors in the pre-trained model, and diff-pruning (Guo et al., 2021), which learns a sparse parameter update vector.

4.1.3 Combining methods

We find empirically that the simple methods we have already introduced are good enough to match full fine-tuning. However, in principle it is possible to combine several methods. One obvious combination is with all methods that can be folded into the original network without changing pre-trained model the architecture. Combining Batch Ensemble, LoRA and Bitfit, we arrive at the following:

$$\vec{h} \leftarrow (\vec{x} \circ \vec{u})\mathbf{W} \circ \vec{v} + s \cdot \vec{x}\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}} + \vec{b}. \tag{4.7}$$

We could set $\mathbf{W} \leftarrow \mathbf{W} \circ \vec{u}\vec{v}^T + s \cdot \mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$ (and if necessary add the bias term for minimal extra inference cost) to leave the architecture unchanged.

In practice, due to the large number of free parameters in this setup (which combination is best, how do we set s, etc.), and the strong empirical performance of other methods, we restrict ourselves to the simpler architectures introduced earlier. Future work might explore this idea further.

4.1.4 Conditional adapters and other methods

In various scenarios we want the modification to hidden states to be conditional on some signal. In other words we want the PEFT method to be a function of a particular input, e.g. the output of some encoder neural network applied to a different mode for multi-modal (e.g. vision + language) applications. PEFT methods are a natural fit, since making the actual weights of the 'parent' network a function of some input requires many parameters. Consider making a weight from the parent model $\mathbf{W} \in \mathbb{R}^{d \times k}$ a simple linear transform of a vector $\vec{x} \in \mathbb{R}^b$: $\text{vec}(\mathbf{W}) = \mathbf{V}\vec{x}$ (assuming we 'flatten' $\mathbf{W}$ to a vector $\text{vec}(\mathbf{W}) \in \mathbb{R}^{dk}$. Now $\mathbf{V}$ requires dkb parameters, b times more than our original network.

We now discuss several methods from the literature, all of which set the parameters of PEFT methods to the output of an encoder function $\text{E}(\vec{x})$, which encodes some 'outside' signal not present in the parent model input. The FiLM approach (Perez et al., 2018) updates hidden states with an affine transformation, i.e. combining $\vec{v}$ and $\vec{b}$ of Equation 4.7. FiLM was introduced as a method of conditioning a model on a new modality, e.g. for visual question answering we could modify the hidden states of a vision model applied to an image with information from a text question. So in this case, $\vec{v}$ and $\vec{b}$ would be the PEFT parameters, conditional on the output of the encoder $\text{E}(\vec{x})$, an LSTM applied to text input.

Karimi Mahabadi et al. (2021a) introduce the 'Hyperformer++' method for multi-task learning in NLP. Here the PEFT parameters are the $\mathbf{W}_{\text{down}}$ and $\mathbf{W}_{\text{up}}$ matrices of adapters and the encoder $\text{E}(\vec{x})$ is a 1-layer FFN applied to the concatenation of layer id embeddings, adapter position embeddings, and 'task' embeddings. In other words, learnable vectors are associated with each layer and adapter position of the parent model, and a learnable vector is associated with each task. The encoder FFN is shared across all layers and tasks, allowing for cross-task and cross-layer information transfer. Similarly Üstün et al. (2020) introduce 'UDadapter' for multilingual dependency parsing. Here again the PEFT parameters are the matrices of adapters and the encoder $\text{E}(\vec{x})$ is a simple linear projection of a language embedding. By learning jointly on many languages we enable cross-lingual transfer.

4.2 Experiments

We now conduct an empirical comparison of the architectures introduced in the previous section. As mentioned previously, we concentrate on the 'frozen backbone' paradigm, with the base model the popular T5 Raffel et al. (2019) family of models. We evaluate the PEFT architectures on accuracy on 5 challenging English text classification tasks from the SuperGLUE and GLUE

Model *large*	Mem. (GB)	Iters. to Conv.	Time (h)	Time to Conv. (h)
ft	18.5	244	2.18	0.56
last3	4.6	585	0.96	0.58
bitfit	8.3	710	1.12	0.83
FFN adapter-1	8.4	956	1.59	1.59
FFN adapter-16	8.7	748	1.16	1.07
FFN adapter-256	6.9	366	1.28	0.58
batch-ens. (all)	12.6	479	1.09	0.65
batch-ens. ($\mathbf{W}_q, \mathbf{W}_1$)	9.8	685	0.94	0.79
batch-ens. ($\mathbf{W}_q, \mathbf{W}_v$)	9.0	809	1.13	1.13
LoRA (all)	10.1	493	1.28	0.99
LoRA ($\mathbf{W}_q, \mathbf{W}_1$)	8.5	444	1.00	0.70
LoRA ($\mathbf{W}_q, \mathbf{W}_v$)	8.2	633	1.05	1.05

Table 4.1: Comparing for a fixed batch size (25) and base model (t5-large): Memory cost, number of iterations to achieve the maximum validation loss, speed of the entire training run of 20 epochs, and training speed normalised by number of iterations to reach convergence ($s_i \cdot \text{it}_i/\max(\text{it}_i)$ for model id i, speed s and number of iterations it). Averaged over three seeds and five medium-sized NLU datasets. Note the training speed for methods other than full FT is artificially low since we could increase batch size and still fit on one GPU.

benchmarks, namely the Recognising Textual Entailment, Corpus of Linguistic Acceptability, Microsoft Research Paraphrase Corpus, Words in Context and Boolq datasets. Performance on each dataset is measured by accuracy unless otherwise stated. From GLUE: **RTE** (Recognizing Textual Entailment) datasets come from a series of annual competitions on textual entailment, formulated as two-class classification: *entailment* and *not_entailment*. **CoLA** (Corpus of Linguistic Acceptability), a dataset of sentences rated by linguistic acceptability, scored by Matthews correlation with human judgements. **MRPC** (Microsoft Research Paraphrase Corpus), a task where you have to classify whether a given sentence is a paraphrase of another one. From SuperGLUE: **WiC** (Word-in-Context, Pilehvar and Camacho-Collados, 2019) is a word sense disambiguation task cast as binary classification of sentence pairs. Given two text snippets and a polysemous word that appears in both sentences, the task is to determine whether the word is used with the same sense in both sentences. **BoolQ** (Boolean Questions, Clark et al., 2019) is a QA task where each example consists of a short passage and a yes/no question about the passage.

The T5 family of models are encoder-decoder transformers, pre-trained on a large and diverse English web corpus, with a 'denoising auto-encoder' objective, with the encoder fed a document with several tokens masked out and the decoder having to reconstruct the original document. Various modern sizes are available, and we tested the models with approximately 60 million, 200 million, 700 million and 3 billion parameters. An 11 billion parameter model is also available, but we did not test this due to compute constraints. We were also unable to make full fine-tuning of the 3 billion parameter model possible with our resources. For more details about T5 and other pre-trained transformers see section 2.3.2.

T5 was chosen due to strong performance on text classification, with subsequent work arguing that the encoder-decoder and denoising auto-encoder framework is close to optimal for fine-tuning performance (as opposed to encoder-only architectures like BERT or decoder-only architectures like GPT-2) Tay et al. (2022b), and we additionally benefit from the range of model sizes available to test with and previous work Mahabadi et al. (2021) using the same model family.

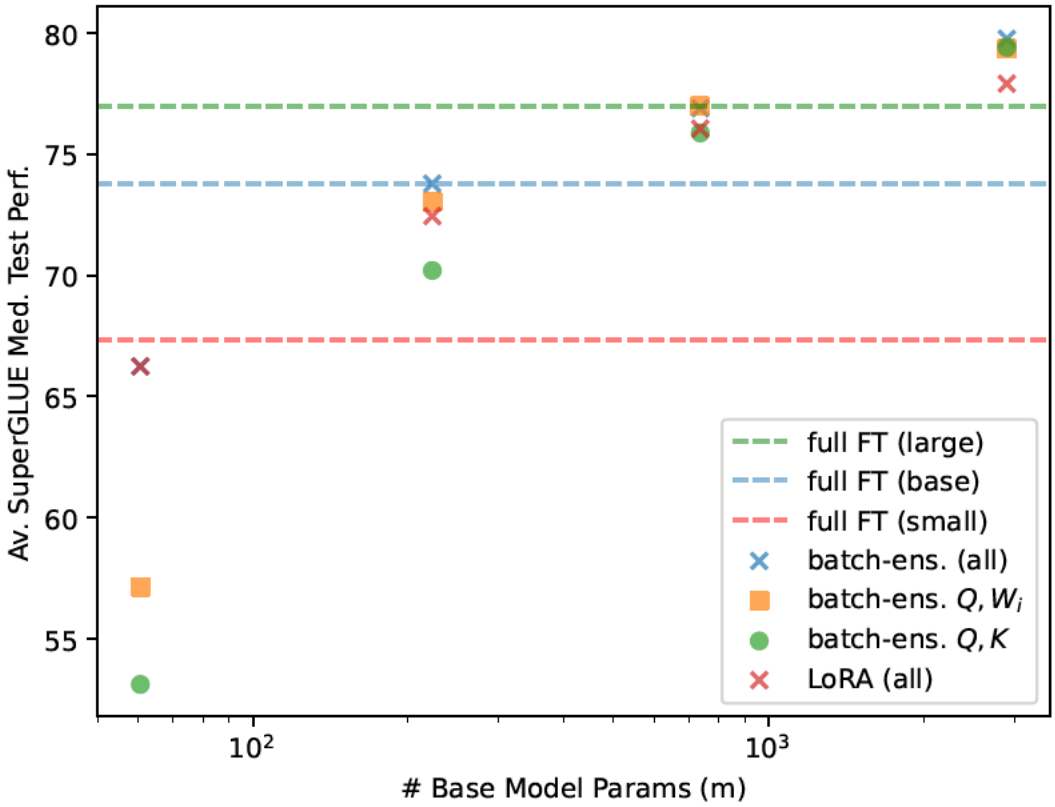


Figure 4.4: Average SuperGLUE performance for different methods across base model size.

4.3 Results and Discussion

We begin by comparing PEFT architectures on their 'selling points' such as reduced memory use and training speed compared to full fine-tuning (section 4.3.1), then discuss their performance on SuperGLUE tasks (section 4.3.3), trends with base model size (section 4.3.4), a discussion of the details of the LoRA method (section 4.3.2) and finally our overall recommendations (section 4.3.3).

4.3.1 Memory usage and convergence speed

A key benefit of PEFT compared to full fine-tuning is reduced GPU memory usage and decreased training time due to updating fewer parameters. In Table 4.1.4 we compare various PEFT architecture choices with respect to these aspects. Firstly, as expected all methods reduce memory costs compared to full fine-tuning ('ft'), in most cases by over 50%. We save the most memory in the case of only tuning the final three layers of the network. This is because we can stop backpropagation after only three layers, whereas other methods require backpropagating through the entire network, even though not every parameter is updated.

Among the other methods there is less variation in memory required. However adding trainable parameters on top of more hidden states, or larger hidden states (i.e. adding LoRA or BatchEnsemble layers in parallel to more network weights) results in more memory required. For

example compare adding LoRA to just the query and value weights (LoRA $(\mathbf{W}_q, \mathbf{W}_v)$) to adding to the query and the first FFN weight[3] (LoRA $(\mathbf{W}_q, \mathbf{W}_1)$) to adding it to every layer (LoRA (all)). A similar pattern holds for BatchEnsemble.

In terms of training speed and convergence, full fine-tuning takes the fewest iterations to converge. Decreasing the number of tuned parameters results in slower convergence, e.g. compare adapters with decreasing bottleneck dimension, or LoRA/BatchEnsemble added to more layers. We can speculate that this is because we have a harder 'search' problem; influencing the frozen model weights to solve the fine-tuning task may be harder with fewer degrees of freedom.

Despite taking more iterations to converge, PEFT methods are faster per-iteration due to not having to calculate gradients for or update most parameters. Every PEFT method is about twice as fast as full fine-tuning. Taking into account the number of iterations to converge, large adapters, BitFit, full fine-tuning and tuning the last three layers are the fastest. However this underestimates the benefits of PEFT methods, since we don't know how many iterations are required to converge ahead of time, and could increase the batch size of PEFT methods to increase training speed, unlike full fine-tuning where we would run out of GPU memory.

4.3.2 Low Rank Layers Ablation

We explore and confirm the necessity of the additions made by Hu et al. (2021) to the 'low-rank layers' architecture introduced in the previous chapter. In particular the output is scaled by g/r, with g a hyperparameter and r the rank of the low-rank decomposition, so the updated hidden states look like $\vec{h} \leftarrow \vec{h} + g/r \cdot \vec{x}\mathbf{W}_{\text{down}}\mathbf{W}_{\text{up}}$. In this section we also consider 'unscaled' LoRA where we simply set $g/r = 1$.

We conduct various ablations, listed in Table 4.3.3. We first confirm that 'unscaled' ($g = r = 1$) LoRA performs poorly, see e.g. LoRA (all, $g = 1$) in Table 4.3.3. We can recover some performance by scaling up the learning rate by 128×, see LoRA (all, $g = 1$, lr=4e-2), but are still far behind full fine-tuning. By testing LoRA methods with various ranks, we found that unscaled rank 128 LoRA performed well, so using $g/128 = 1$ implies $g = 128$, and we confirm that transferring this value of g to rank 1 or rank 4 LoRA results in good performance, e.g. LoRA (all) almost recovers full fine-tuning performance.

4.3.3 Performance of different methods

Batch ensemble mostly outperforms rank-1 LoRA, especially when adapting every linear transformation (methods marked 'all'); see Table 4.3.3. Exceptions are T5-base for LoRA $(\mathbf{W}_q, \mathbf{W}_v)$ and T5-3b for LoRA $(\mathbf{W}_q, \mathbf{W}_1)$. On the other hand, BatchEnsemble consistently requires more memory than LoRA, see Table 4.1.4; as noted before, modifying extra hidden states leads to more memory usage, and BatchEnsemble modifies (with a Hadamard product) hidden states before and after a linear transformation, whereas LoRA only modifies hidden states *after* a linear transformation.

Tuning the last three layers performs the worst overall, and additionally has the highest parameter count. However it does require less memory and computation than other methods. Of methods adapting every layer, BitFit typically performs the worst (or close to it), and medium-sized adapters (FFN adapter-16) perform well, outperforming all methods for t5-large and t5-3b.

Putting this together, we can offer the following recommendations:

- If you mostly care about saving memory, and are not worried about performance, you should only fine-tune the last three layers. If this method is the only way to 'upgrade' to

[3]The first FFN weight $\mathbf{W}_1 \in \mathbb{R}^{d \times 4d}$ results in hidden states four times larger than value weights $\mathbf{W}_v \in \mathbb{R}^{d \times d}$.

a bigger model size, then this will probably give you better performance than a different PEFT method applied to a smaller model size.

- If you mostly care about retaining the original model architecture exactly, and don't want to fine-tune many parameters, you should probably use BatchEnsemble, which will normally give you better performance than the other methods which leave the new architecture unchanged, LoRA, BitFit or fine-tuning the last three layers. However it seems to be worth exploring both LoRA and BatchEnsemble, and we can increase the rank of LoRA, leading to more capacity, with no obvious way to do the same for BatchEnsemble. Additionally, when easy vectorisation is important, for example adapting to several different tasks in one batch, BatchEnsemble is a useful method.
- If you mostly care about performance, you should use adapters. They are reasonably competitive on memory usage, doing better than LoRA or BatchEnsemble, and lead to the best performance overall. One downside is the architecture will be changed, and inference will be slightly slower due to having to pass hidden states through the adapter layers as well as the original model layers.

4.3.4 Trends with model size

On a high level, we see two trends as base model size increases: the gap between full fine-tuning and PEFT decreases, and the difference between methods decreases; see Figure 4.4 for a visual illustration with BatchEnsemble and LoRA. This perhaps indicates that we should put greater emphasis on factors such as simplicity or memory cost rather than just performance, since it seems likely that as model scale continues to increase, any PEFT architecture will be able to match full fine-tuning.

For every model size, even the worst performing PEFT method with a larger base model outperforms full fine-tuning a smaller base model. For example FFN adapter-1 with T5-3b as the base model (270k parameters) outperforms tuning T5-large (738 million parameters). So if you have a fixed GPU memory budget it should be worth using PEFT methods to increase the base model size while staying within your budget. Though of course using a larger model size imposes higher inference costs, and PEFT methods do nothing to reduce those.

4.3.5 Conclusions and Limitations

Overall we can conclude that the adapter models used prominently in this thesis have many advantages, with strong performance across model sizes and fairly low memory cost. We have also showed that PEFT methods perform well with increasing model scale, and provided some recommendations of which method to use for different scenarios. The key distinctions between methods are how much they modify the original architecture, their memory cost, and the overall performance on downstream tasks.

We can roughly split methods into the following taxonomy:

- **Simple, least memory, worst performance**: BitFit, fine-tuning the last three layers. These methods are particularly simple to implement, with the only change to typical training code being optimizing a subset of model parameters. They also save the most on memory costs, but suffer from lower performance than other methods.
- **Close to original architecture, medium performance, most memory cost**: LoRA, BatchEnsemble. These methods have fairly strong performance, and can leave the original model architecture unchanged, but are slightly more complicated to implement, and require the most memory.

Model	% Tuned	No. Tuned (m)	RTE	COLA	MRPC	WIC	BoolQ	Average
small								
ft		60.49	63.9	43.0	84.9	66.8	74.4	66.6
bitfit	0.19%	0.12	58.0	11.1	84.2	62.5	68.2	56.8
FFN adapter-1	0.06%	0.03	60.6	0.3	84.8	63.3	68.6	55.5
FFN adapter-1; g=128	0.06%	0.03	64.1	30.9	86.8	63.3	71.4	63.3
base								
ft		222.88	76.9	55.9	88.2	68.4	79.2	73.7
last3	12.69%	28.3	68.4	53.9	86.8	62.5	77.1	69.7
bitfit	0.13%	0.28	71.7	50.1	87.8	65.9	76.1	70.3
FFN adapter-1; g=128	0.05%	0.10	75.3	54.4	87.3	65.7	77.7	72.1
FFN adapter-1	0.05%	0.10	71.8	47.5	89.5	66.8	76.9	70.5
FFN adapter-16	0.29%	0.66	74.6	54.7	88.2	67.1	78.7	72.7
FFN adapter-256	4.27%	9.51	76.9	58.9	86.8	65.9	79.3	73.5
batch-ens. (all)	0.18%	0.41	77.3	58.1	87.7	66.8	79.0	73.8
batch-ens. ($\mathbf{W}_q, \mathbf{W}_1$)	0.07%	0.15	73.4	59.1	87.6	66.9	78.3	73.1
batch-ens. ($\mathbf{W}_q, \mathbf{W}_v$)	0.05%	0.11	69.3	50.7	87.8	65.6	77.6	70.2
LoRA (all)	0.18%	0.41	74.7	54.8	87.7	67.2	77.7	72.4
LoRA ($\mathbf{W}_q, \mathbf{W}_1$)	0.07%	0.15	73.8	55.2	88.3	66.8	78.3	72.5
LoRA ($\mathbf{W}_q, \mathbf{W}_v$)	0.05%	0.11	75.1	49.1	88.3	66.5	76.7	71.2
large								
ft		737.64	81.2	58.8	90.1	72.3	82.6	77.0
last3	6.82%	50.3	80.7	57.5	88.3	67.5	80.3	74.9
bitfit	0.09%	0.70	80.6	58.3	89.5	66.9	80.4	75.2
FFN adapter-1; g=128	0.04%	0.27	82.3	58.4	89.5	69.5	82.4	76.4
FFN adapter-1	0.04%	0.27	80.7	57.7	90.0	70.1	81.4	76.0
FFN adapter-16	0.24%	1.75	83.6	59.2	91.2	70.4	82.7	77.4
FFN adapter-256	3.44%	25.35	83.5	59.3	89.2	70.0	83.3	77.1
batch-ens. (all)	0.15%	1.08	82.8	59.4	90.4	69.0	82.9	76.9
batch-ens. ($\mathbf{W}_q, \mathbf{W}_1$)	0.05%	0.39	82.8	58.8	90.6	70.4	82.2	77.0
batch-ens. ($\mathbf{W}_q, \mathbf{W}_v$)	0.04%	0.29	80.4	58.3	90.8	68.3	81.6	75.9
LoRA (all)	0.15%	1.08	81.6	55.8	90.7	70.2	82.0	76.1
LoRA ($\mathbf{W}_q, \mathbf{W}_1$)	0.05%	0.39	81.6	54.8	90.5	69.0	81.7	75.5
LoRA ($\mathbf{W}_q, \mathbf{W}_v$)	0.04%	0.29	82.1	56.0	90.0	69.2	81.0	75.7
3b								
bitfit	0.06%	1.95	83.2	65.2	90.3	72.7	84.6	79.2
FFN adapter-1	0.01%	0.27	83.8	62.5	89.8	71.2	85.1	78.5
FFN adapter-16	0.06%	1.75	83.5	66.4	90.8	73.6	85.1	79.9
batch-ens. (all)	0.10%	3.15	82.6	65.8	89.7	75.2	85.5	79.8
batch-ens. ($\mathbf{W}_q, \mathbf{W}_1$)	0.04%	1.20	82.7	65.7	89.3	73.0	86.1	79.3
batch-ens. ($\mathbf{W}_q, \mathbf{W}_k$)	0.02%	0.74	83.2	64.7	91.3	72.5	85.3	79.4
LoRA (all)	0.10%	3.15	81.2	66.8	90.9	72.3	85.4	79.3
LoRA ($\mathbf{W}_q, \mathbf{W}_1$)	0.04%	1.20	83.2	65.3	89.8	74.7	84.9	79.6
LoRA ($\mathbf{W}_q, \mathbf{W}_v$)	0.02%	0.74	83.0	62.1	89.1	71.2	84.1	77.9

Table 4.2: Performance of various PEFT methods on medium-sized NLU datasets. 'ft' refers to full fine-tuning, 'last3' refers to only tuning the final three layers of the decoder, 'bitfit' refers to only tuning the bias parameters, 'FFN adapter-d' refers to adapters with bottleneck dimension d, and $g = 128$ refers to multiplying the adapter output by 128 like in LoRA, 'batch-ens. (x)' refers to BatchEnsemble applied to layers x and 'LoRA (x)' refers to applying rank-1 LoRA updates to layers x. Accuracies are averaged over three seeds.

			RTE	COLA	MRPC	WIC	BoolQ	Average
small								
ft		60.49	63.9	43.0	84.9	66.8	74.4	66.6
batch-ens. (all)	0.22%	0.14	67.0	38.1	87.3	65.2	73.6	66.2
batch-ens. ($\mathbf{W}_q, \mathbf{W}_1$)	0.08%	0.05	62.0	4.7	84.8	63.5	70.8	57.1
batch-ens. ($\mathbf{W}_q, \mathbf{W}_k$)	0.06%	0.04	54.8	1.5	82.8	61.9	64.6	53.1
LoRA (all)	0.22%	0.14	64.9	42.3	86.2	65.0	72.9	66.2
LoRA ($\mathbf{W}_q, \mathbf{W}_1$)	0.08%	0.05	63.8	36.7	85.2	63.5	71.2	64.1
LoRA ($\mathbf{W}_q, \mathbf{W}_v$)	0.06%	0.04	56.9	0.8	84.3	63.4	67.9	54.7
LoRA (all, g=1)	0.22%	0.14	56.9	-0.3	72.3	53.4	64.9	49.4
LoRA ($\mathbf{W}_q, \mathbf{W}_k$; g=1)	0.06%	0.04	54.3	0.7	69.0	50.1	62.0	47.2
LoRA (all; g=1; lr=4e-2)		0.14	63.4	12.4	84.8	63.3	65.4	57.9
LoRA ($\mathbf{W}_q, \mathbf{W}_k$; g=1; lr=4e-2)		0.04	59.0	0.7	80.2	56.4	62.2	51.7
LoRA (all; r=4)	0.89%	0.54	65.3	43.5	86.4	65.4	73.6	66.8
LoRA ($\mathbf{W}_q, \mathbf{W}_v$; r=4)	0.24%	0.15	61.4	0.9	85.1	61.7	70.4	55.9

Table 4.3: Comparing batch ensemble and LoRA with various ablations, with t5-small as the base model: 1) not scaling the LoRA output layer (g=1), 2) increasing learning rate by the same amount the output is scaled (lr=4e-2), 3) increasing LoRA rank (r=4).

- **Best performance, most complicated, further from original architecture**: adapters. Slightly more complicated to implement than the previous methods due to including a non-linearity, and increase inference costs slightly due to adding 'extra' layers, but offer the best performance.

We did not evaluate all PEFT methods from the literature, avoiding methods like prefix-tuning (Li and Liang, 2021) due to their lower performance and higher memory costs (explored further in Chapter 5, Section 5.1), and did not explore all variants of the adapter architecture, but tried to cover the methods that tend to get used in practice. We also did not test on decoder-only models, focusing on the T5 architecture due to it being the best available model that had a wide range of sizes available at the time of performing these experiments.

Part II

Parameter-efficient methods for machine translation

Chapter 5

Domain adaptation and transfer learning for MT

As discussed in previous chapters, a promising paradigm for transfer learning is to freeze most of a pre-trained model trained on some large corpus with an unsupervised objective, and only tune a small number of parameters on some downstream task, e.g. sentiment analysis or natural language inference. We discuss relevant design choices and performance as pre-trained model size increases in Chapter 4. This chapter explores extending this 'frozen backbone' approach from English classification tasks to machine translation (MT).

Machine translation involves automatically translating text from one language to another. The language that is translated *from* is known as the source language and the language translated *to* is known as the target language. Variants of MT include 'English-centric' MT, where either the source or target language is English, 'multilingual MT', where one model handles multiple translation directions (e.g. French to English and German to English) and 'many-to-many' MT, where all possible translation directions for n languages are handled by one model (e.g. French to and from English, German to and from English, French to and from German).

We present two papers in this chapter. The first is focused on adapting models not trained on MT to the MT task, adapting either to a single language pair (where English was either the source or target language) and (English-centric) multilingual MT (section 5.1). The second is focused on adapting a model already trained on MT to a specific domain such as legal text (section 5.4), focusing on many-to-many multilingual translation.

Various considerations are different for the MT task compared to classification. The MT task is in some sense more complicated, requiring us to estimate the probability distribution over sequences of text in multiple languages, rather than just estimating the probability that a piece of text has a particular label. Both papers presented in this chapter discuss aspects of this difference, with techniques originally introduced for the classification setting no longer working. MT is also inherently multilingual, and we presumably need a multilingual pre-trained model to achieve the best performance. We contrast adapting an English-only model and a multilingual model in the first paper presented in this chapter.

Finally, for many language pairs there are millions or even billions of parallel sentences (i.e. a translation of the same sentence into two languages) available to train on. Do we really need 'transfer learning' for such cases, or should we just directly train on the task of interest? We test our techniques on both lower resource language pairs where there is on the order of 100k parallel sentences as well as higher resource language pairs where there are millions of parallel sentences

in the first paper presented in this chapter.

5.1 Recipes for adapting pre-trained monolingual and multilingual models to machine translation

This section presents the paper *Recipes for Adapting Pre-trained Monolingual and Multilingual Models to Machine Translation* (Stickland et al., 2021b), the paper was initially published as a preprint on arXiv in April 2020. Then, it was accepted for publication at the conference *European chapter of the Association for Computational Linguistics (EACL)* in April 2021.

This paper investigates the benefits and drawbacks of freezing parameters, and adding adapters, when fine-tuning a pre-trained model on MT. We focus on 1) Fine-tuning a model trained only on English monolingual data. 2) Fine-tuning a model trained on monolingual data from 25 languages. Neither model was trained on machine translation specifically, but both are sequence-to-sequence transformers with an encoder and decoder, trained on a 'denoising autoencoder' objective, which involved recovering the original version of a corrupted piece of text. These corruptions were simple operations like masking words and permuting the order of sentences in a document.

Our main contribution was exploring for the first time the effect of freezing pre-trained model parameters when fine-tuning on machine translation. Our key findings were:

- We could effectively use an English-only pre-trained sequence-to-sequence model for machine translation, by introducing a new encoder that feeds into the pre-trained encoder.
- For multilingual pre-trained models, we showed the importance of fine-tuning cross-attention parameters, with the intuition that we already have good representations in both the source and target language and need to align them using cross-attention.
- The MT task requires more parameters to be fine-tuned to match full fine-tuning than the English classification tasks we considered before, over 10% of total parameters vs. the less than 1% that we observed in previous chapters, see further discussion in Chapter 6.
- For multilingual pre-trained models, we showed that it is important to freeze the encoder when English is the source language, and important to freeze the decoder when English is the target language. This is likely because English is overrepresented in the pre-training data, so the representations for English are good enough already without any fine-tuning.
- For smaller datasets with noisy training data and English as the target language, we showed we could outperform full fine-tuning with our best freezing strategies.

We did not find any benefit from freezing parameters for higher resource languages with lots of data. But we could get limited benefits for lower resource languages. However we did not conduct any control experiments where we only vary dataset size, or only vary language pair, in order to isolate the effect of each one on performance. In Chapter 6 we present such experiments, with essentially the same experimental setup.

All the experiments in this paper had English as either the source or target language, and in section 5.4 we cover many-to-many multilingual translation (albeit with a different pre-trained model and setup).

Author contributions

Asa Cooper Stickland wrote code for the paper, carried out most of the experiments, and largely wrote the paper. Xian Li provided feedback and guidance throughout the project and paper-writing, and carried out the multilingual MT experiments. Marjan Ghazvininejad provided initial code for adapting BART to MT, and also provided feedback on the project and paper.

5.2 The paper

Recipes for Adapting Pre-trained Monolingual and Multilingual Models to Machine Translation

Asa Cooper Stickland[♣] **Xian Li**[♠] **Marjan Ghazvininejad**[♠]

[♣] University of Edinburgh, [♠] Facebook AI
, {

Abstract

There has been recent success in pre-training on monolingual data and fine-tuning on Machine Translation (MT), but it remains unclear how to best leverage a pre-trained model for a given MT task. This paper investigates the benefits and drawbacks of freezing parameters, and adding new ones, when fine-tuning a pre-trained model on MT. We focus on 1) Fine-tuning a model trained only on English monolingual data, BART. 2) Fine-tuning a model trained on monolingual data from 25 languages, mBART. For BART we get the best performance by freezing most of the model parameters, and adding extra positional embeddings. For mBART we match or outperform the performance of naive fine-tuning for most language pairs with the encoder, and most of the decoder, frozen. The encoder-decoder attention parameters are most important to fine-tune. When constraining ourselves to an out-of-domain training set for Vietnamese to English we see the largest improvements over the fine-tuning baseline.

1 Introduction

Machine Translation (MT) has recently seen significant advances, with improvements in modeling, especially since the advent of neural models (Sutskever et al., 2014; Bahdanau et al., 2015), and the availability of large parallel corpora for training such systems (Smith et al., 2013; Kocmi and Bojar, 2017; Tiedemann, 2012). However, often standard neural systems do not perform well on *low-resource* language pairs (Koehn and Knowles, 2017), especially when the language pairs are only distantly related. Since these languages are spoken by a large fraction of the world's population, reducing the gap in performance between high and low-resource MT could have a large impact.

An explosion of interest in large-scale pre-training in Natural Language Processing has led to increased performance on smaller datasets, by simple *fine-tuning* of large pre-trained models on downstream tasks. The typical approach is to train a large model on text from the web (for example English Wikipedia), with a common objective predicting masked out tokens using the unmasked context. For Natural Language Generation (for example summarization of text), performance can be improved by pre-training a sequence-to-sequence model (Song et al., 2019; Lewis et al., 2019).

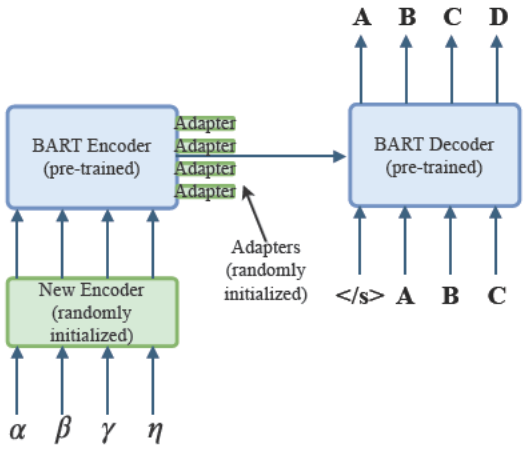


Figure 1: Schematic diagram showing the components of our system for adapting BART to MT. We learn a new encoder that takes as input the source language, with a potentially different vocabulary to the original BART system. We freeze most BART parameters (frozen model components are shown in blue).

However previous work has shown that on NLP tasks such as Natural Language Inference, the relative performance of fine-tuning vs. keeping the pre-trained model frozen depends on the similarity of the pre-training and downstream tasks (Peters et al., 2019). We observe empirically that simple fine-tuning of a monolingual model for MT can result in worse performance than training from scratch (e.g. Table 1). For MT the more common mono-

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 3440–3453
April 19 - 23, 2021.

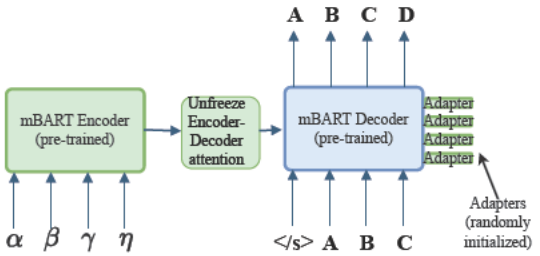


Figure 2: Schematic diagram showing one method of adapting mBART to MT, unfreezing the encoder and encoder-decoder attention, and adding adapters in the decoder. Model components colored blue are not updated during fine-tuning.

lingual (usually only English) pre-training (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019; Yang et al., 2019b; Liu et al., 2019) may be inadequate since the input or output domain for the downstream task will be a non-English language.

Multilingual pre-training offers a solution, by modifying the pre-training objective to include many languages. Using a multilingual pre-trained model for MT gives good performance, especially on lower-resource language directions (Liu et al., 2020). However it is challenging to balance the training data so that higher-resource languages do not overwhelm lower-resource ones (Arivazhagan et al., 2019; Conneau et al., 2019). For a particular language it may be hard to source monolingual data, or it may be simply not included in training.

We also consider multilingual MT (training on many language pairs and sharing all or most model parameters) as a downstream task. Sharing 'knowledge' across language directions can improve performance on low-resource language pairs by transfer from other pairs included in training. Previous work observed problems of performance degradation, often on high-resource languages, due to interference and constrained capacity (Johnson et al., 2017; Tan et al., 2019). And when initialising from a pre-trained model, we want to avoid 'catastrophic forgetting', where by fine-tuning on a particular language pair we lose the knowledge about another language pair that is stored in the model weights.

Previous work has explored how to improve on simple fine-tuning, by freezing pre-trained model parameters (Peters et al., 2019; Houlsby et al., 2019) and using lightweight 'adapter modules' (Houlsby et al., 2019; Stickland and Murray, 2019) which are inserted between the layers of the pre-trained network. We aim to explore and improve on these approaches for both bilingual and multilingual MT (in contrast to previous work largely focusing on text classification). We explore freezing different subsections of the pre-trained model.We expect freezing to be particularly useful when the parallel data is of low quality, in which case naive fine-tuning may, for example, over-specify the pre-trained model to a particular domain.

Our main contributions are:

- A novel fine-tuning approach, similiar to Lewis et al. (2019) but with adapter modules in the encoder of the pre-trained sequence-to-sequence model and combining both learnable, and fixed sinusoidal, positional embeddings in the input module (see sections 3.1 and 3.2) that feeds into the pre-trained encoder.

- Extensive experiments with fine-tuning a multilingual pre-trained model for MT, showing the benefits and drawbacks of freezing various parameters. We find we should freeze the decoder but unfreeze the encoder-decoder attention when fine-tuning on $Xx \rightarrow En$ data, and in the other direction we should freeze the encoder but unfreeze the entire decoder (section 5.3). We find monolingual models benefit more from freezing parameters than multilingual models (section 5.2).

- Results on fine-tuning a multilingual pre-trained model for multilingual MT showing that freezing parameters improves performance on some, mostly distantly related, language directions (section 5.5).

2 Background and Related Work

BART and mBART We briefly describe the pre-trained models we focus on in this work. In order to perform machine translation with the minimum of modifications to the pre-trained model, we prefer models that can perform conditional sequence generation. We concentrate on the BART (Bidirectional and Auto-Regressive Transformer) model (Lewis et al., 2019) and the multilingual BART (mBART; Liu et al., 2020) model. BART and mBART are sequence-to-sequence models with the standard transformer-based neural machine translation architecture, i.e. an encoder and autoregressive decoder. The pre-training task they are trained on is reconstructing a document from

a noisy version of that document (so called 'denoising autoencoder'). Examples of noise added to the training data include randomly shuffling the order of the original sentences, randomly changing the start position of the document, and using a masking scheme where arbitrary length spans of text are replaced with a single mask token. BART and mBART are trained entirely on monolingual data from the web, with English data for BART and data from 25 different languages for mBART.

BART and mBART have almost identical architectures, with 12 encoder layers and 12 decoder layers with model dimension of 1024 and 16 attention heads. BART has a vocabulary of approximately 40k and $\sim$ 406M parameters, whereas mBART has a larger vocabulary of size 250k and $\sim$ 610M parameters.

Pre-trained Models for MT There has been much recent progress in pre-training for NLP applications (Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019; Yang et al., 2019b; Liu et al., 2019), with the most relevant for our work focusing on text generation (Radford et al., 2019; Song et al., 2019; Dong et al., 2019; Raffel et al., 2019; Lewis et al., 2019) Specifically for MT, Ramachandran et al. (2017) proposed pre-training the encoder-decoder modules as two separate language models, and Yang et al. (2019a); Zhu et al. (2020) explored approaches incorporating BERT model weights into the usual seq-to-seq architecture.

Multilingual MT *Multilingual translation* (Firat et al., 2016; Viégas et al., 2016; Aharoni et al., 2019; Arivazhagan et al., 2019) aims to jointly train one translation model that translates multiple language directions, and shares representations to improve the translation performance on low-resource languages (Gu et al., 2018). Our freezing approach is similar in spirit to Sachan and Neubig (2018) who investigate which parameters are most useful to share for multilingual MT with transformer models. We start from a multilingual pre-trained model, and decide between sharing or freezing parameters.

Transfer Learning for MT *Transfer learning* hopes to leverage a related task to perform well on a target task, for example by initialising the model weights from those resulting from training on a related task. For MT various approaches have been explored, with a common method training on high-resource language(s) and fine-tuning on a low-resource language (Neubig and Hu, 2018).

Closely related to our work is that of Bapna and Firat (2019), who introduce freezing and adapters (extra parameters inserted within the transformer) for domain adaption in MT. They take an MT model trained on a large parallel corpus, and fine-tune in a different domain (e.g. legal text). We differ in that we start from a pre-trained model that has not been trained on parallel text, and study adapting it to MT. Approaches based on freezing various model components have also been proposed (Thompson et al., 2018; Zoph et al., 2016), but have focused on RNN models pre-trained with parallel data, not transformer models pre-trained on monolingual data.

3 Methods

Because BART has been trained on only English input, we need to use different techniques when fine-tuning BART and mBART for MT, with a schematic overview shown in Figure 1 and Figure 2. BART and mBART are standard sequence-to-sequence models, where an *encoder* consumes a sequence of source-side tokens, and a *decoder* acts as a conditional language model, generating target tokens given a source sequence. Intuitively, we want the encoder and decoder to be performing roughly the same tasks during fine-tuning as they were during pre-training. For BART this means the input to the encoder should be similar to (embedding vectors of) noisy English text. Therefore when training on say, Vietnamese to English, we first transform the Vietnamese source sentence into a representation useful for BART. We introduce new parameters (the 'Input Module') that consume the source sentence and produce hidden vectors we can feed into the BART encoder. We describe the Input Module architecture in section 3.1.

mBART can be fine-tuned without modification since during pre-training it saw the languages it will be fine-tuned on. To increase flexibility when freezing parts of the network, we optionally add extra parameters to both BART and mBART, described in section 3.3.

3.1 Input Module Architecture

We refer to the network that takes in the source language text and outputs hidden vectors useful for BART as an 'Input Module' or $\mathrm{IM}(\cdot)$. To improve performance on low-resource MT, we use smaller token embedding vectors on the source side of size $d_{\mathrm{s}} = 512$, whereas BART uses hidden vectors of

size $d_{\text{BART}} = 1024$. The full network is as follows, with $\{\mathbf{e}_t\}_{t=0}^{l}$ token embeddings for a source sentence with l tokens,

$$\text{BART}(\text{IM}(\{\mathbf{e}_t\}_{t=0}^{l})), \tag{1}$$

where $\text{BART}(\cdot)$ is the full BART encoder-decoder model. Where we would normally input token embeddings to the BART model we use the outputs of the Input Module. The t-th element of $\text{IM}(\{\mathbf{e}_t\}_{t=0}^{l})$ as follows:

$$\alpha\text{LN}(\mathbf{W}\text{Transformer}(\{\mathbf{e}_t\}_{t=0}^{l})_t) \tag{2}$$

and where $\text{LN}(\cdot)$ is layer-norm, $\mathbf{W}$ is a matrix projecting up from d_{s} to d_{BART}, and $\text{Transformer}(\cdot)$ is the application of a series of Transformer layers. α is a scalar, in our case equal to $\sqrt{d_{\text{BART}}}$, which is required to insure the input to BART is on the same scale as the embedding vectors BART was trained on. If we remove $\text{LN}(\cdot)$, $\mathbf{W}$ and α, and set $d_{\text{s}} = d_{\text{BART}}$, we recover the method introduced by Lewis et al. (2019) for fine-tuning BART on MT.

3.2 Extra Positional Embeddings

We found empirically that the details of positional embedding vectors are important for good performance (see Table 1), perhaps because of the need for the BART model to deal with different word order to that it was trained on. Transformer models normally have either learnable positional embedding vectors, or fixed sinusoidal positional embedding (Vaswani et al., 2017) vectors $\mathbf{p}^t$, with $\mathbf{p}^t_i = \sin(t/10000^{i/(d_s/2-1)})$, if $0 \leq i < d_s/2$, and $\mathbf{p}^t_i = \cos(t/10000^{(i-(d_s/2-1))/(d_s/2-1)})$ if $d_s/2 \leq i < d_s$, where t indexes position and i indexes dimension.

Note that positional embedding are typically only added to the token embeddings. We use learnable positional embeddings at the embedding layer. But to get extra positional information, we optionally add fixed sinusoidal positional embedding to the input of each transformer layer in $\text{IM}(\cdot)$, i.e. the input to layer i, $\mathbf{h}^i_t = \mathbf{o}^{i-1}_t + \mathbf{p}^t$, with $\mathbf{o}^{i-1}_t$ the previous layer output. This means the network has access to both *learned positional embeddings* (only at the embedding layer), and *fixed sinusoidal* ones at the input to each layer.

3.3 Within-Network Adapter Architecture

When freezing parts of a pre-trained model (either BART or mBART in our case), we may want to add flexibility by modifying the pre-trained model architecture. One approach is to use 'adapters', introduced by Houlsby et al. (2019); Stickland and Murray (2019) which are newly-initialised neural network layers that can be 'slotted in' to the layers of the pre-trained model.

We only considered simple adapter architectures, essentially feed-forward networks, with one hidden layer, and a residual connection to the output. The dimension of the hidden layer can be much smaller than the model dimension to reduce computational cost and parameter count. We use one adapter per transformer layer, inserting them at the end of the layer (Stickland and Murray, 2019; Bapna and Firat, 2019). We use the following architectures, with $\mathbf{h}$ the hidden state of a particular token after the usual transformer layer, and $\mathbf{h}_{\text{out}}$ the hidden state of the token after the adapter layer:

$$\begin{aligned} \mathbf{z} &= \text{gelu}(\mathbf{W}_d\mathbf{h}) \\ \mathbf{h}_{\text{out}} &= \tanh(\mathbf{W}_u\mathbf{z}) + \mathbf{h} \end{aligned} \tag{3}$$

The $\tanh$ non-linearity helped with stability in early experiments, probably because it prevents the adapter output exploding by constraining it between -1 and 1.

We also considered a version of the adapter based on the 'gated linear unit' (GLU; Dauphin et al., 2016) architecture:

$$\begin{aligned} \mathbf{z} &= 2\sigma(\mathbf{W}_g\mathbf{h}) \odot \text{gelu}(\mathbf{W}_d\mathbf{h}) \\ \mathbf{h}_{\text{out}} &= \tanh(\mathbf{W}_u\mathbf{z}) + \mathbf{h}. \end{aligned} \tag{4}$$

We found the network was sensitive to changes in the magnitude of the hidden states the adapter produced, and therefore multiply the sigmoid gate by 2 so that it approximately leaves the magnitude of the hidden states unchanged.

3.4 Freezing Details

BART We freeze all parameters of BART except the weights and biases of the layer-norm modules (following Houlsby et al. (2019)), and additionally unfreeze the self-attention module of the first layer in the BART encoder, which is a small fraction of total BART parameters ($24 \cdot 2d_{\text{BART}}$ from layer-norm parameters and $4d^2_{\text{BART}}$ from the self-attention module). We freeze BART token embeddings (used in the softmax layer).

mBART In most of our experiments we unfreeze layer-norm parameters, positional and token embeddings, and either the entire encoder or decoder

Languages	Vi-En	Tuned Params (m)
(1): BART + InputModule (unfreeze all)	9.5	374
(2): BART (frozen) + InputModule	27.9	26
(3): (2) + unfreeze layer-norm	28.4	26
(3) + sinusoidal positional embeddings	18.3	26
(1) + extra positional embeddings	22.0	26
(4): (3) + extra positional embeddings	29.0	26
(5): (3) + encoder adapters	28.9	29
(3) + decoder adapters	28.3	29
(6): (5) + extra positional embeddings	30.0	29
(7): (6) + GLU adapters	30.5	29

Table 1: Ablation study for various choices in the frozen BART method, with validation set BLEU score. We organise model settings by a number in brackets, (n), and define a new model configuration in bold as **(n)**:. We use '+' to indicate the addition of new model settings on top of the previous ones. Method **(2)** is similar to the method introduced by Lewis et al. (2019). '+ sinusoidal positional embeddings' refers to adding sinusoidal positional embeddings to token embeddings, while '+ extra positional embeddings' refers to adding them within each transformer layer (see section 3.2). 'Tuned Params (m)' refers to the number of tunable parameters for each method in millions. Test set results are listed in Table 3 (as 'Frozen BART').

Languages	It-En	Si-En
(1): BART + InputModule + LN	34.1	5.1
(2): (1) + encoder adapters	35.0	7.3
(1) + decoder adapters	35.5	6.8
(3): (2) + extra pos. embeddings	36.3	8.7
(4): (3) + GLU adapters	35.7	9.2

Table 2: Further Ablation study for key settings of the frozen BART method, with validation set BLEU score. Test set results are listed in Table 3 (as 'Frozen BART').

module (or the encoder and subsections of the decoder). We unfreeze the self-attention module of the first layer in the mBART encoder and decoder.

4 Experimental Settings

We use the fairseq (Ott et al., 2019) library for all experiments. The final models are selected based on validation likelihood, except for multilingual fine-tuning where we evaluate the models after 10000 training steps. We use beam-search with beam size 5 for decoding, and evaluate all BLEU scores using SacreBLEU (Post, 2018) [1]. We use ISO 693-2 language codes in this work for convenience, and use the same parallel data as Liu et al. (2020), both listed in listed in Table 11 of the Appendix.

We fine-tune frozen BART and an Input Module on bilingual parallel text, feeding the source language into the Input Module. For mBART we feed the source language into the encoder, and use the same hyper-parameters as Liu et al. (2020). When using adapters we use 0.1 dropout in the adapter bottleneck layer ($\mathbf{z}$ in section 3.3), and a hidden dimension of either 128, or $\lfloor 2/3 \cdot 128 \rceil$ when using a gated linear unit adapter. We use the Adam (Kingma and Ba, 2015) optimizer. Hyper-parameters are listed in Appendix B, and we use the same hyper-parameter search space for frozen and non-frozen models.

4.1 Multilingual MT

We train with a very large effective batch size, training on 32 GPUs with a per-GPU batch size of 4096 tokens, meaning our total batch size is $N \cdot 32 \cdot 4096$ tokens, where N is the number of language pairs. We evaluate our model after 10000 training steps (amounting to $N \cdot 10000$ forwards-backwards passes through the model).

4.2 Vocabulary

BART uses the GPT-2 tokenizer, which uses the BPE (Sennrich et al., 2016) approach (on the level of bytes, not characters). BART could technically take any Unicode string as input, however the BPE is learned on English text. When fine-tuning BART on machine translation we therefore learn a

[1] SacreBLEU signature: `BLEU+case.lc+lang.[src-lang]-[tgt-lang]+numrefs.1+smooth.exp+tok.13a+version.1.3.6`

Languages **Size**	**Vi-En**† 110k	**Vi-En** 133k	**It-En** 250k	**My-En** 259k	**Ne-En** 564k	**Si-En** 647k	**Cs-En** 11M	**Es-En** 15M	**Pars (m)**
(1): Freeze decoder	12.1	30.0	36.5	27.4	11.0	13.6	26.6	34.1	407
Freeze encoder	12.0	29.7	36.6	25.2	8.8	12.3	25.6	33.8	457
(2): (1) + adapters	12.2	30.0	36.7	27.7	10.8	14.2	27.4	34.4	410
(2) + ft enc-attn	12.3	30.6	37.0	29.0	11.4	14.9	27.0	35.1	461
(2) + ft self-attn	11.7	30.4	36.1	28.3	10.6	14.3	27.4	34.7	461
(2) + ft last 3 lyrs	12.1	30.6	36.6	28.1	11.5	14.7	27.6	34.9	461
Test (random init)	8.1	23.6	31.7	23.3	7.6	7.2	22.0	29.0	N/A
Test (frozen BART)	-	35.2	38.5	21.0	0.5	7.8	-	-	29
Test (ft all)	14.1	**36.7**	**39.8**	**27.6**	14.1	**14.0**	29.2	**34.5**	610
Test (ft enc-attn)	**14.9**	**36.4**	39.4	**27.9**	**14.6**	**14.1**	**29.8**	**34.4**	461

Table 3: Validation BLEU score (unless stated otherwise) obtained by freezing various parts of the mBART and of adding adapters for *Xx* → *En*. 'ft' refers to fine-tuning, i.e. unfreezing. Vi-En† refers to a new parallel, 'out-of-domain' dataset constructed similarly to the Flores (Guzmán et al., 2019) train sets (see section 5.2). 'Test (frozen BART)' indicates results from English-only BART with the best performing method from Table 2 or Table 1. 'Test (random init)' refers to training models (of various sizes) from scratch on the bitext for that language pair. 'Pars (m)' refers to the number of tunable parameters for each method in millions (note token embeddings are tuned in every method and account for 256m parameters). Bold indicates the best test set score and all scores whose difference from the best is not statistically significant (with p-value less than 0.05). (Statistical significance is computed via bootstrapping (Koehn, 2004).)

	Vi-En	**It-En**	**My-En**	**Ne-En**	**Si-En**
Freeze decoder (don't ft layer-norm)	26.6	35.1	26.6	10.3	13.1
Freeze encoder (don't ft layer-norm)	29.4	36.1	24.1	8.7	12.1

Table 4: Ablation study on improvement from fine-tuning layer-norm. Compare to the 'Freeze decoder' and 'Freeze encoder' methods in the first two rows of Table 3.

new subword vocabulary (using the sentencepiece (Kudo and Richardson, 2018) library) on the source data from the fine-tuning dataset, and use a smaller vocabulary size of 5000, which empirically performs better for low-resource MT (Guzmán et al., 2019; Sennrich and Zhang, 2019). We don't change the mBART tokenizer or vocabulary.

5 Results and Discussion

5.1 Frozen BART

Table 1 shows the effects of various choices we made in fine-tuning BART for MT. *Freezing* is important: we see an 18.4 BLEU point improvement from fine-tuning a frozen BART model compared to fine-tuning an unfrozen BART (both with an Input Module; see section 3.1).

Adding extra flexibility with within-network adapters helps performance, especially when added to the BART *encoder*. It is important to use *learned positional embeddings* at the embedding layer in the Input Module, with an 10.1 BLEU score drop if we use fixed positional embeddings (at the embedding layer). We see consistent gains in Table 1 and Table 2 by adding additional, fixed sinusoidal positional embeddings to the input of every transformer layer of the Input Module (see section 3.2), even when using an unfrozen BART. The BART encoder 'expects' English input, and it may be the Input Module with extra fixed embeddings can better account for the different word order in the input language. In the next section we compare to mBART and baselines.

5.2 Frozen mBART

In Table 3 and Table 5 we list results from freezing various parts of mBART. We get better performance than fine-tuning ('ft all' in Table 3) with our freeze decoder + fine-tune encoder-decoder attention method ('ft enc-attn' in Table 3) on Ne-En and Cs-En for *Xx* → *En*, and mostly similar results to

Languages	En-Vi	En-It	En-My	En-Ne	En-Si	En-Cs	En-Es	Pars (m)
Freeze decoder	29.7	32.2	35.0	5.8	2.1	17.7	35.4	407
(1): Freeze encoder	30.1	31.5	36.0	5.3	3.7	16.5	35.0	457
(2): (1) + encoder adapters	30.3	32.3	36.9	5.4	4.2	16.6	35.3	461
Test (ft all)	**35.4**	34.0	**36.9**	**7.4**	**3.3**	**18.0**	**34.0**	610
Test (freeze enc. + adapters)	35.0	**34.3**	35.9	6.9	**3.3**	16.7	32.5	461

Table 5: Validation BLEU score (unless stated otherwise) obtained by freezing various parts of the mBART and of adding adapters for for *En* → *Xx*. 'Pars (m)' refers to the number of tunable parameters for each method in millions.

the baseline otherwise.

We believe a benefit to freezing, when fine-tuning on training data from a different domain to test data, will be avoiding specialising the pre-trained model to the fine-tuning train data domain. To test this we constructed a new Vi-En parallel dataset (Vi-En† in Table 3) using the some of the same sources as the Flores (Guzmán et al., 2019) training data (the Si-En and Ne-En training sets used in this work), specifically GNOME/KDE/Ubuntu domain from the OPUS repository[2] and Bible translations from the bible-corpus[3], and use the same test and validation sets as the IWSLT15 Vi-En dataset. By constraining ourselves to this out-of-domain training set we see the largest gains out of the language pairs we considered over the fine-tuning baseline (0.9 BLEU).

We also consider the effect of the size of the fine-tuning dataset. If we constrain the training data to a random subset of 200k training examples from Ro-En (Table 6), the 'ft enc-attn' method outperforms simple fine-tuning. This effect generalises to an mBART variant that was pre-trained on only Ro and En monolingual data (using the same data as Liu et al. (2020)). Further results on Ro-En data are available in the Appendix, Table 10, and show similar trends to Table 3, with fine-tuning encoder-decoder attention the most important.

Table 3 shows the relative performance of frozen BART, frozen mBART and baselines. Fine-tuning mBART gave consistently better results than frozen BART especially for distantly related languages. For Si, Ne and My the performance of frozen BART is roughly on par with a randomly initialised model (or much worse in the case of Ne-En). The parallel data for these languages is often lower quality, and the BART system has to learn about the non-English language from noisy or out-of-domain text (e.g. text from the Ubuntu manual for the En-Ne pair). For Vi and It, we have high quality parallel data, and the frozen BART method is only approximately 1.5 BLEU points behind the best mBART results. We note mBART was trained on more English data than BART, and with different noising function hyper-parameters.

5.3 What Should be Unfrozen?

Layer-Norm We find large benefits to simply fine-tuning the weights and biases of the pre-trained layer-norm weights (recall that after normalisation, the layer-norm module multiplies each hidden dimension by a weight and adds a bias); this was observed in the setting of BERT by Houlsby et al. (2019). This gains e.g. 0.5 BLEU for frozen BART (see Table 1) and an average of 0.8 BLEU across five languages for mBART (see Table 4 compared to Table 3). Since these weights and biases are only $2d$ parameters per layer-norm, where d is the model dimension. This is parameter-efficient, with adding more parameters with 'Adapters' on top of unfrozen layer-norm providing a smaller improvement.

Encoder vs Decoder For the *Xx* → *En* direction (Table 3) we can see that freezing the decoder always performs better than freezing the encoder (except for It-En where they perform roughly the same.) For the *En* → *Xx* direction (Table 5) we see slightly weaker evidence for the opposite trend, with the decoder more useful to fine-tune; but for the high resource languages Es and Cs freezing the decoder works better. There is more English data in mBART pre-training than data in other languages, which may account for better results with a frozen encoder (when English is the source language) or decoder (when English is the target language). Adding flexibility with adapters in the frozen layers

[2] http://opus.nlpl.eu/

[3] https://github.com/christos-c/bible-corpus/

Model	mBART		En-Ro mBART	
Languages (Size)	**Ro-En (608k)**	**Ro-En (200k)**	**Ro-En (608k)**	**Ro-En (200k)**
Test (ft all)	37.8	36.4	38.5	37.7
Test (ft enc-attn)	37.8	36.8	38.1	37.9

Table 6: Validation set BLEU (unless stated otherwise) comparing freezing various parts of mBART and En-Ro mBART (pre-trained only on En and Ro data), fine-tuned on *Ro* $\rightarrow$ *En* parallel data. 'ft' referes to fine-tuning, i.e. unfreezing. 'Ro-En (200k)' refers to a random subset of the Ro-En training data of size 200k.

Src. Lang.	**Ru**	**Fr**	**De**	**Zh**	**Es**	**Cs**	**Lv**	**Fi**	**Lt**	**Et**	**Hi**	**Si**
Size	32M	29M	28M	25M	15M	11M	4.5M	2.7M	2.1M	1.9M	788k	647k
Finetune all	**33.6**	**39.0**	**33.1**	**20.2**	33.7	**29.9**	**21.1**	**29.0**	**22.8**	**28.6**	**25.4**	**16.9**
Ft enc-attn	**33.4**	38.2	32.6	**20.2**	**34.0**	29.7	**20.8**	**29.1**	**22.7**	**28.3**	25.1	**16.7**
Src. Lang.	**Ro**	**Ne**	**My**	**Ar**	**It**	**Nl**	**Ko**	**Ja**	**Tr**	**Vi**	**Kk**	**Gu**
Size	612k	563k	259k	251k	251k	237k	230k	223k	207k	133k	91k	12k
Finetune all	**37.8**	**20.7**	**31.0**	**37.0**	**39.6**	**43.3**	**25.0**	**18.7**	**24.0**	**37.4**	14.6	**18.7**
Ft enc-attn	**37.9**	**20.8**	30.5	**36.9**	**39.3**	**43.0**	24.2	**18.8**	23.7	**37.5**	**15.0**	**18.3**

Table 7: Test set BLEU score on many-to-one (*Xx* $\rightarrow$ *En*) multilingual MT with a simple round-robin training schedule. 'Ft enc-attn' refers to fine-tuning the encoder, and fine-tuning the encoder-decoder attention module in every decoder layer, leaving the other decoder sub-modules frozen. The 'Ft enc-attn' model setting uses adapter modules in the decoder to increase flexibility after freezing parameters. Bold indicates the best score and all scores whose difference from the best is not statistically significant (with p-value less than 0.05). For clarity we underline language pairs where the 'Ft enc-attn' method matches or outperforms naive fine-tuning.

improves performance in all languages and directions, except for Ne$\rightarrow$En.

We explore more fine-grained unfreezing for the *Xx* $\rightarrow$ *En* direction (Table 3). We fine-tuned three equally sized subsets of the decoder: the encoder-decoder attention layers (approx. $12 \cdot 4d^2_{\text{BART}}$ parameters), the self-attention layers in the decoder (approx. $12 \cdot 4d^2_{\text{BART}}$ parameters), or the entire last three layers of the decoder (approx. $3 \cdot 16d^2_{\text{BART}}$ parameters). We observe that fine-tuning the encoder-decoder attention performed well (note the last three layers include three encoder-decoder attention layers), with fine-tuning self-attention the least useful. We hypothesize that the pre-training task of mBART (reconstructing noisy monolingual sentences) does not help with teaching the encoder-decoder attention to align source and target text of different languages.

5.4 Memory Cost

Freezing parameters means we no longer need to allocate memory to storing their gradients. We will obtain additional memory savings when using an optimizer that stores various other quantities (i.e. the Adam optimizer stores running averages of the first and second moments of gradients.). The memory savings allow for roughly 45-75% larger batches for the methods we consider in this work (see Table 8 for our mBART methods), but for larger pre-trained models the proportion of GPU memory freed up by freezing will increase. At inference time we no longer require gradients and we have the same memory cost.

	Tokens per GPU
Finetune all	2304
(1): Freeze decoder	4096
Freeze encoder	3584
(2): (1) + decoder adapters	4096
(2) + ft enc-attn	3328

Table 8: Maximum number of tokens that would fit on one NVIDIA Volta GPU when fine-tuning mBART on the En-Vi training set. We evaluated batch sizes in increments of 256 tokens.

5.5 Multilingual Fine-tuning of mBART

We explore freezing parts of the mBART model when fine-tuning on a challenging multilingual MT

task. Table 7 lists results from a naive fine-tuning baseline, and results from freezing most of the decoder but unfreezing the encoder-decoder attention (when freezing we use GLU adapters in the decoder, see section 3.3). Freezing parameters hurts performance on some language pairs, and since freezing removes flexibility from the model and we have to adapt to 25 different directions this is perhaps not surprising. The language pairs where we match or improve on the baseline are Zh, Es, Fi, Ne, Ja, Vi and Kk. These are mostly (five out of seven) non-European languages, and distantly related to En. However since most of these results are not statistically significant further study is needed to verify this. Note we see a clear benefit over bilingual fine-tuning for some language pairs (e.g. compare our best Ne result from Table 3, 14.6 BLEU vs. 20.8 BLEU for multilingual fine-tuning). We leave to future work a more thorough investigation of the multilingual MT setting.

6 Conclusion

We recommend: For a language with high quality parallel data but without a pre-trained model trained on monolingual data from that language, using a frozen (English-only) BART model with additional parameters at the source side (the 'input module') improves performance over a randomly initialised baseline. For this approach it is important to freeze the pre-trained model. We also give the model both learned positional embeddings at the embedding layer, and fixed sinusoidal positional embeddings at each layer of the input module.

For a multilingual pre-trained model, we found performance improvements on some (mostly distantly related) languages for multilingual many-to-one fine-tuning. For bilingual $En \rightarrow Xx$ fine-tuning we did not see any improvement, although the performance drops are small, and by freezing parameters we need less memory at training time compared to fine-tuning. For $Xx \rightarrow En$ bilingual fine-tuning it is important to unfreeze the encoder-decoder attention, and keep the rest of the decoder frozen. This can improve on simple fine-tuning, especially for distantly-related language pairs or those with out-of-domain training data.

We recommend fine-tuning layer-norm parameters as a parameter-efficient complement to adapter layers. For our mBART experiments we found it was necessary to fine-tune the token embeddings, which correspond to a large number of parameters, and future work could remove this cost by working out a subset of the vocabulary to fine-tune, or another method.

Acknowledgments

We'd like to thank James Cross, Mike Lewis, Naman Goyal, Jiatao Gu, Iain Murray, Yuqing Tang and Luke Zettlemoyer for useful discussion. We also thank our colleagues at FAIR and FAIAR for valuable feedback.

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *CoRR*, abs/1907.05019.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Unsupervised cross-lingual representation learning at scale.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2016. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of

deep bidirectional transformers for language understanding. In *North American Association for Computational Linguistics (NAACL)*.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *arXiv preprint arXiv:1905.03197*.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *NAACL*.

Jiatao Gu, Hany Hassan, Jacob Devlin, and Victor O.K. Li. 2018. Universal neural machine translation for extremely low resource languages. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 344–354, New Orleans, Louisiana. Association for Computational Linguistics.

Francisco Guzmán, Peng-Jen Chen, Myle Ott, Juan Pino, Guillaume Lample, Philipp Koehn, Vishrav Chaudhary, and Marc'Aurelio Ranzato. 2019. The FLORES evaluation datasets for low-resource machine translation: Nepali–English and Sinhala–English. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6097–6110, Hong Kong, China. Association for Computational Linguistics.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799, Long Beach, California, USA. PMLR.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Tom Kocmi and Ondřej Bojar. 2017. Curriculum learning and minibatch bucketing in neural machine translation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 379–386, Varna, Bulgaria. INCOMA Ltd.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 875–880, Brussels, Belgium. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. FAIRSEQ: A fast, extensible toolkit for sequence modeling. In *North American Association for Computational Linguistics (NAACL): System Demonstrations*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *North American Association for Computational Linguistics (NAACL)*.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. *CoRR*, abs/1903.05987.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Time Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, OpenAI.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Prajit Ramachandran, Peter J Liu, and Quoc Le. 2017. Unsupervised pretraining for sequence to sequence learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 383–391.

Devendra Sachan and Graham Neubig. 2018. Parameter sharing methods for multilingual self-attentional translation models. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 261–271, Belgium, Brussels. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Association for Computational Linguistics (ACL)*, pages 1715–1725.

Rico Sennrich and Biao Zhang. 2019. Revisiting low-resource neural machine translation: A case study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy. Association for Computational Linguistics.

Jason R. Smith, Herve Saint-Amand, Magdalena Plamada, Philipp Koehn, Chris Callison-Burch, and Adam Lopez. 2013. Dirt cheap web-scale parallel text from the common crawl. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1374–1383, Sofia, Bulgaria. Association for Computational Linguistics.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: Masked sequence to sequence pre-training for language generation. In *International Conference on Machine Learning (ICML)*.

Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected attention layers for efficient adaptation in multi-task learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5986–5995, Long Beach, California, USA. PMLR.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Xu Tan, Jiale Chen, Di He, Yingce Xia, Tao Qin, and Tie-Yan Liu. 2019. Multilingual neural machine translation with language clustering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 963–973, Hong Kong, China. Association for Computational Linguistics.

Brian Thompson, Huda Khayrallah, Antonios Anastasopoulos, Arya D. McCarthy, Kevin Duh, Rebecca Marvin, Paul McNamee, Jeremy Gwinnup, Tim Anderson, and Philipp Koehn. 2018. Freezing subnetworks to analyze domain adaptation in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 124–132, Belgium, Brussels. Association for Computational Linguistics.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2214–2218, Istanbul, Turkey. European Languages Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Fernanda Viégas, Greg Corrado, Jeffrey Dean, Macduff Hughes, Martin Wattenberg, Maxim Krikun, Melvin Johnson, Mike Schuster, Nikhil Thorat, Quoc V Le, et al. 2016. Google's multilingual neural machine translation system: Enabling zero-shot translation.

Jiacheng Yang, Mingxuan Wang, Hao Zhou, Chengqi Zhao, Yong Yu, Weinan Zhang, and Lei Li. 2019a. Towards making the most of bert in neural machine translation. *arXiv preprint arXiv:1908.05672*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019b. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. 2020. Incorporating bert into neural machine translation. *arXiv preprint arXiv:2002.06823*.

Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. 2016. Transfer learning for low-resource neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1568–1575, Austin, Texas. Association for Computational Linguistics.

A Additional Ablation Study

In Table 9 we reproduce Table 4 of the main paper with more context to study the effect of unfreezing layer-norm parameters when fine-tuning mBART. Across all language pairs we see improvements from fine-tuning layer norm parameters over not fine-tuning them, and additional, smaller, improvements from adding adapters, indicating both forms of adding flexibility are useful. In Table 10 we present additional results on the Ro-En pre-trained model (see section 3.2 of the main body).

B Fine-tuning Hyper-parameters

For all experiments with bilingual datasets we use a batch size of 2048×16 tokens, i.e. 2048 tokens per GPU and 16 GPUs (we investigate larger batch sizes for frozen models only to test GPU memory usage, and do not evaluate models trained with larger batch sizes). Ranking of hyper-parameters was done by validation set BLEU score.

Frozen BART We train with 0.3 dropout for the frozen BART parameters, and 0.2 dropout for the Input Module parameters, 0.1 label smoothing, 0.2 dropout for the self-attention scores in the Input Module, 5000 warm-up steps, and $7\mathrm{e}{-4}$ maximum learning rate. We performed a grid search over learning rates in $\{7\mathrm{e}{-4}, 5\mathrm{e}{-4}, 3\mathrm{e}{-4}\}$, dropout for Input Module parameters in $\{0.2, 0.1\}$, and dropout for self-attention scores in $\{0.2, 0.1\}$. We train for a maximum of 50K training updates for all low and medium resource pairs and 100K for high resource pairs (which takes roughly 8 hours and 16 hours respectively).

Frozen mBART We train with 0.3 dropout, 0.2 label smoothing, 2500 warm-up steps, and $3\mathrm{e}{-5}$ maximum learning rate. We did not search over hyper-parameters, simply re-using those of Liu et al. (2020). Despite the adapter parameters being randomly initialised, the small learning rate did not affect performance (we performed a small sweep of larger learning rates and found only marginal gains, and so kept the same settings for simplicity). We use a maximum of 40K training updates for all low and medium resource pairs and 100K for high resource pairs (Es and Cs in our case), this takes roughly 12 hours and 30 hours respectively.

Multi-lingual MT We train with 0.3 dropout, 0.1 dropout for self-attention scores, 4000 warm-up steps, and $1\mathrm{e}{-4}$ maximum learning rate.

Out-of-domain Vi-En Baseline To train a randomly initialised baseline for the out-of-domain Vi-En data (Vi-En† in Table 3 of the main body) we used the same model architecture and training settings as those of Guzmán et al. (2019) use for training MT systems on similar data (but with Si or Ne source language). Specifically a seq2seq transformer with 5 encoder and decoder layers, hidden dimension 512. shared embeddings between the input and softmax layers, and strong regularisation (e.g. 0.4 dropout on hidden states, 0.2 dropout on attention scores, 0.2 label smoothing). We learn a BPE vocabulary (joint across source and target data) of size 5000 on the training data. For full details of hyper-parameters we refer the reader to Guzmán et al. (2019) and the associated GitHub repository[4].

C Pre-training Languages

We reproduce in Table 11 the details from Liu et al. (2020) of the size of each pre-training language corpus for mBART.

[4]https://github.com/facebookresearch/flores

	Vi-En	It-En	My-En	Ne-En	Si-En
Freeze decoder	26.6	35.1	26.6	10.3	13.1
Freeze encoder	29.4	36.1	24.1	8.7	12.1
(1): Freeze decoder + ft layer norm	30.0	36.5	27.4	11.0	13.6
Freeze encoder + ft layer norm	29.7	36.6	25.2	8.8	12.3
(1) + decoder adapters	30.0	36.7	27.2	10.8	14.2

Table 9: Validation BLEU score (unless stated otherwise) obtained by fine-tuning layer-norm parameters and of adding adapters for mBART, for *Xx* $\rightarrow$ *En*. 'ft' refers to fine-tuning, i.e. unfreezing. Note we are simply reproducing rows from Table 3 and Table 4 of the main paper for ease of comparison.

	mBART		En-Ro mBART	
	Ro-En (608k)	Ro-En (200k)	Ro-En (608k)	Ro-En (200k)
(1): Freeze decoder	38.8	37.9	40.4	39.9
Freeze encoder	39.1	38.3	40.0	39.2
(2): (1) + decoder adapters	39.3	38.0	40.6	40.0
(1) + ft enc-attn	39.8	39.0	40.5	40.5
(1) + ft self-attn	39.6	38.3	40.4	40.1
(1) + ft last 3 lyrs	39.6	38.6	40.5	40.3
Test (ft enc-dec)	37.8	36.8	38.1	37.9
Test (ft all)	37.8	36.4	38.5	37.7

Table 10: Validation set BLEU (unless stated otherwise) comparing freezing various parts of mBART and En-Ro mBART (pre-trained only on En and Ro data rather than 25 languages), fine-tuned on *Ro* $\rightarrow$ *En* parallel data. 'ft' refers to fine-tuning, i.e. unfreezing. 'Ro-En (200k)' refers to a random subset of the Ro-En training data of size 200k.

Code	Language	Tokens(M)	Size(GB)	Parallel data source
En	English	55608	300.8	
Ru	Russian	23408	278.0	WMT19
Vi	Vietnamese	24757	137.3	IWSLT15
Ja	Japanese	530 (*)	69.3	IWSLT17
De	German	10297	66.6	WMT19
Ro	Romanian	10354	61.4	WMT16
Fr	French	9780	56.8	WMT19
Fi	Finnish	6730	54.3	WMT17
Ko	Korean	5644	54.2	IWSLT17
Es	Spanish	9374	53.3	WMT19
Zh	Chinese (Sim)	259 (*)	46.9	WMT19
It	Italian	4983	30.2	IWSLT17
Nl	Dutch	5025	29.3	IWSLT17
Ar	Arabic	2869	28.0	IWSLT17
Tr	Turkish	2736	20.9	IWSLT17
Hi	Hindi	1715	20.2	ITTB
Cs	Czech	2498	16.3	WMT19
Lt	Lithuanian	1835	13.7	WMT19
Lv	Latvian	1198	8.8	WMT17
Kk	Kazakh	476	6.4	WMT19
Et	Estonian	843	6.1	WMT18
Ne	Nepali	237	3.8	FLoRes
Si	Sinhala	243	3.6	FLoRes
Gu	Gujarati	140	1.9	WMT19
My	Burmese	56	1.6	WAT19

Table 11: **Languages and Statistics of the CC25 Corpus.** A list of the 25 languages used in mBART pretraining ranked with monolingual corpus size. (*) The Chinese and Japanese corpora are not segmented, so the token counts here are sentence counts.

5.3 Contribution and impact

This paper has 22 citations according to Google Scholar as of 2023-04-04.

In our work we fed an encoding of a non-English source sentence from a randomly initialized transformer encoder into (English-only) BART. One could generalize this to the general idea of freezing a pre-trained model trained on a particular domain, and feeding in representations from another domain, allowing us to use powerful representations from the frozen pre-trained model while incorporating new sources of information. Tran et al. (2020) build on this idea; they use both text-only and speech-only pre-trained models for the task of speech-to-text translation. Their approach uses a pretrained encoder from wav2vec 2.0 (Baevski et al., 2020) for acoustic modeling and a pre-trained decoder from mBART (Liu et al., 2020) for language modeling. Then, similar to our work, they freeze most pre-trained model parameters, only training newly initialized layers to convert the output of the speech encoder into suitable input for the language decoder (they use a stack of several 1-dimensional convolutional layers with stride 2 to shrink the speech sequence (encoder output) by a factor of 2^n). Similar to our case, they observe improved cross-lingual generalization by only fine-tuning layer norm and attention parameters rather than fine-tuning the whole model. Le et al. (2021) experiment with a very similar scenario, using a different pre-trained speech encoder and the decoder from mBART, and again find freezing most parameters and only training cross-attention and adapter modules results in the best performance.

Gheini et al. (2021) further explores our finding that cross-attention parameters are particularly important to fine-tune, obtaining parity with full fine-tuning. Their work explores the related, but different setting of adapting a model trained on just two languages to a third (e.g. from French to English to Romanian to English).

Several papers extend the general paradigm of parameter-efficient fine-tuning for MT. Philip et al. (2020) introduce independent source and target language adapters in the encoder and decoder respectively, for the purposes of multilingual translation (e.g. a German source adapter and a French target adapter are activated for German to French translation). Chronopoulou et al. (2022) introduce *language family* adapters instead of adapters per-language, for the purposes of sharing information from high resource to low resource languages in the same family.

So far we have mostly discussed the task of adapting a multilingual pre-trained model that was not trained on machine translation to MT. However we might worry that this is not a practical scenario. After all, it is possible to find large amounts of parallel data for many language pairs, why not train on that data instead of using an unsupervised objective? The next section assumes we have such a pre-trained machine translation model, and that we want to adapt it to a particular specialized domain, such as legal or medical text.

5.4 Multilingual Domain Adaptation for NMT: Decoupling Language and Domain Information with Adapters

This section presents the paper *Multilingual Domain Adaptation for NMT: Decoupling Language and Domain Information with Adapters* (Cooper Stickland et al., 2021). The paper was initially published as a preprint on arXiv in October 2021. It was accepted for publication at the conference *The Sixth Conference in Machine Translation (WMT21)* in November 2021.

In this paper we explore various aspects of parameter-efficient *domain adaptation* for machine translation. Previous chapters and the previous sections of this chapter focus on a scenario where we have a pre-trained model and want to adapt it to a task that is very different from the pre-training objective. For example going from a masked language modeling (e.g. BERT) or denoising objective (e.g. T5 or mBART) to tasks like classification or machine translation. In this section we instead focus on a scenario where there is a large overlap between pre-training

and fine-tuning. We assume the objective for both is machine translation, but the pre-training objective is in a 'generic domain' such as text scraped from the web, whereas the fine-tuning objective is in a specific domain, in our case text from legal, medical, IT or religious sources. Such specific domains will have their own vocabulary, semantics, and common phrases or sentence structures. For example the word 'server' will appear in very different contexts if you are talking about technology vs. an American restaurant.

In this work we are specifically interested in *multilingual* domain adaptation. We consider both a *full* resource scenario where we can train on domain-specific data in many languages (say English, French and German), and test on all of them, and also a *partial* resource scenario where we train on domain-specific data in only a subset of the languages we test on (say train on French and test on German). In the partial resource scenario we are interested in the cross-lingual transfer performance of various methods. Previous studies have shown it is possible to combine language-specific (as opposed to language-pair specific) adapters (Philip et al., 2020), or language and task adapters (Pfeiffer et al., 2020) trained independently, enabling zero-shot compositions of adapters. We are interested in the question of whether it is possible to combine both language-specific and domain-specific adapters to improve cross-lingual transfer for a particular domain. More concretely can we combine at inference time adapters for source language x, target language y and domain z that were never seen together at training time?

Our baseline model is smaller than the ones we considered in previous sections and chapters, both in terms of parameter count and amount of training data. We use a Transformer Base (Vaswani et al., 2017) trained on English-centric (e.g. English to and from German/French are represented, but not French to German) ParaCrawl v7.1 data (Esplà et al., 2019) with 12 relatively high-resource European languages (803M line pairs in total). ParaCrawl is based off a large scale crawl of the web, and therefore contains diverse text that we consider to be a 'generic' domain.

As mentioned previously, we consider two scenarios. The 'full resource' scenario consists of fine-tuning the pre-trained model on four domains with all language directions represented (a multilingual, many-to-many setup). In the full resource scenario our key findings were:

- We find that encoder-only adapters can be just as effective as default adapters added in every layer.
- Composing domain adapters with language adapters outperforms language adapters alone, and both approaches outperform naive fine-tuning.
- Fine-tuning all model parameters with domain tags performs the best for most domains.

The strength of the domain tag approach confirms the power of the 'task-specific parameters' approach of Chapter 3, since 'domain tags' are simply domain-specific tokens prepended to each sentence (i.e. a learnable vector of parameters like in prefix/prompt tuning). More generally we can make the following analogies to the Chapter 3 setup: GLUE task $\longrightarrow$ domain + language direction, PALs/adapters $\longrightarrow$ domain tags, BERT $\longrightarrow$ the transformer trained on ParaCrawl. Testing different variants of task-specific parameters would have been interesting but we stuck to domain tags due to its popularity as an existing baseline for domain adaptation in machine translation, and our focus on cross-lingual transfer.

We also considered a 'partial resource' scenario where we test the ability of our model to generalize from a subset of the pre-training languages to all pre-training languages. This was the primary focus of our work. Our key findings were:

- Adding domain-specific adapters only in the decoder, not the encoder, improves cross-lingual transfer in the case where the target language was seen at fine-tuning time but the source language was unseen (e.g. train on English and German legal text, test on French $\longrightarrow$ German legal text). The intuition here is that the encoder doesn't 'specialize' to particular languages, resulting in better generalization to new source languages.

- Regularisation by randomly skipping the domain adapter results in better cross-lingual transfer due to more independence between languages and domains.
- Cross-lingual transfer in the case where the target language was not seen at fine-tuning time (e.g. train on English and German legal text, test on German $\longrightarrow$ French legal text) is difficult for all of our approaches. A persistent problem is 'of-target' translation where we translate into the wrong language (say, translating into English instead of French, if we didn't see French training data).
- A simple fix to this problem is data augmentation with English-centric back-translation. This works even better with the aforementioned decoder-only domain adapters, perhaps because it avoids specializing the encoder to the model-generated English source sentences.

We note the problem of not being able to generalize to new target languages is in contrast to the case of multilingual classification, where composing language and task adapters worked well (Pfeiffer et al., 2020). This is in line with previous sections of this chapter, where we found tuning large parts of the pre-trained model (e.g. cross-attention parameters) was needed to match full fine-tuning for machine translation, whereas for classification we only need to tune lightweight adapters (or even just bias parameters).

Author contributions

Asa Cooper Stickland, wrote the code for and carried out most of the experiments, and largely wrote the paper. Alexandre Bérard provided feedback throughout the project and paper-writing, and implemented the code and ran the experiments for the domain tags setting. Vassilina Nikoulina helped come up with the initial idea and set the direction of the project as well as giving feedback throughout and helping to write the paper.

5.5 The paper

Multilingual Domain Adaptation for NMT: Decoupling Language and Domain Information with Adapters

Asa Cooper Stickland* University of Edinburgh
Alexandre Bérard **Vassilina Nikoulina**
NAVER LABS Europe
first.last@naverlabs.com

Abstract

Adapter layers are lightweight, learnable units inserted between transformer layers. Recent work explores using such layers for neural machine translation (NMT), to adapt pre-trained models to new domains or language pairs, training only a small set of parameters for each new setting (language pair or domain). In this work we study the compositionality of language and domain adapters in the context of Machine Translation. We aim to study, 1) parameter-efficient adaptation to multiple domains and languages simultaneously (full-resource scenario) and 2) cross-lingual transfer in domains where parallel data is unavailable for certain language pairs (partial-resource scenario). We find that in the partial resource scenario a naive combination of domain-specific and language-specific adapters often results in 'catastrophic forgetting' of the missing languages. We study other ways to combine the adapters to alleviate this issue and maximize cross-lingual transfer. With our best adapter combinations, we obtain improvements of 3-4 BLEU on average for source languages that do not have in-domain data. For target languages without in-domain data, we achieve a similar improvement by combining adapters with back-translation. Supplementary material is available at https://tinyurl.com/r66stbxj.

1 Introduction

Multilingual Neural Machine Translation (NMT) has made a lot of progress recently (Johnson et al., 2017; Bapna and Firat, 2019; Aharoni et al., 2019; Zhang et al., 2020; Fan et al., 2020a) and is now widely adopted by the community and MT service providers. Multilingual NMT models handle multiple language directions at once and allow for knowledge transfer to low-resource languages. Machine translation systems often need to be adapted to specific domains like legal or medical text. However, when adapting multilingual systems, in-domain data for most language pairs might not exist. We would like to be able to leverage data in a subset of language pairs to transfer domain knowledge to other languages.

Straightforward methods of domain adaptation include fine-tuning (Freitag and Al-Onaizan, 2016) or usage of domain tags (Kobus et al., 2017; Britz et al., 2017) for different domains. For these methods each new domain request would require retraining the whole model, which is a costly procedure. And naive training on a subset of languages typically reduces performance on all other languages (Garcia et al., 2021), a phenomenon known as 'catastrophic forgetting' (McCloskey and Cohen, 1989).

An alternative technique for adapting such models to new language-pairs or domains are 'adapter layers' (Bapna and Firat, 2019), lightweight, learnable units inserted between transformer layers. A previously trained large multilingual model can be adapted to each language-pair by learning only these small units, and keeping the rest of the model frozen. This procedure also allows for the incremental adding of new language pairs and/or domains to the pre-trained model, reducing the cost of adaptation. Previous studies have shown it is possible to combine language-specific (as opposed to language-pair specific) adapters (Philip et al., 2020), or language and task adapters (Pfeiffer et al., 2020) trained independently, enabling zero-shot compositions of adapters. Our ultimate goal is, for ease of deployment and storage, a single model that can handle all languages and domains. In this work we analyse how to combine *language adapters* with *domain adapters* in multilingual NMT, and study to what extent the domain knowledge can be transferred across languages.

First, we show it is hard to decouple language

*Work done during an internship at NAVER LABS Europe.

knowledge from domain knowledge when fine-tuning multilingual MT systems on new domains. In Section 5.2 we demonstrate that adapters learnt on a subset of language pairs fail to generate into languages not in that subset. Such generation into the wrong language is referred to as 'off-target' translation. We additionally find combinations of domain and language adapters not seen at training time lead to bad performance. We examine how adapter placement and other techniques can improve the compositionality of language and domain adapters when dealing with source or target languages that do not have in-domain data (which we refer to throughout this work as "**out-of-domain languages**"). Our key contributions are:

- We examine domain adaptation capacity in the multi-lingual, multi-domain setting. We find that encoder-only adapters can be just as effective as default adapters added in every layer, and that composing domain adapters with language adapters outperforms language adapters alone, although fine-tuning with domain tags performs better for most domains.
- We improve the cross-lingual transfer of domain knowledge for adapters. We analyse different language and domain adapter combinations that improve performance and reduce off-target translations. Our best results for translation into out-of-domain languages use decoder-only domain adapters, regularisation with domain adapter dropout, and data augmentation with English-centric back-translation.

2 Related Work

Cross-lingual transfer Many works have demonstrated that large pre-trained multilingual models (Devlin et al., 2019; Conneau et al., 2020; Liu et al., 2020) fine-tuned on high-resource languages (or language pairs) can transfer to lower-resource languages in various tasks: Natural Language Inference (Conneau et al., 2018), Question Answering (Clark et al., 2020), Named Entity Recognition (Pires et al., 2019; K et al., 2020), Neural Machine Translation (Liu et al., 2020) and others (Hu et al., 2020).

Domain adaptation in NMT Domain adaptation has been discussed extensively for bilingual NMT models. A typical approach is to fine-tune a model trained on a large corpus of 'generic' data on a smaller in-domain corpus (Luong and Manning, 2015; Neubig and Hu, 2018). A common technique to make use of monolingual in-domain data is to do back-translation (Sennrich et al., 2016a; Berard et al., 2019a; Jin et al., 2020). Although effective, it is expensive to create back-translated data, especially when one needs to cover multiple language pairs. Multi-domain models can be trained with domain tags (Kobus et al., 2017; Britz et al., 2017; Berard et al., 2019a; Stergiadis et al., 2021) that can encode domain-specific information. However, domain tags do not allow *incrementally* adding new domains to a model: each new domain adaptation requires retraining the full model (as opposed to adapter layers that can be trained independently for each language/domain). There are a number of works (Jiang et al., 2020; Britz et al., 2017; Dabre et al., 2020) trying to explicitly decouple domain-specific representations from domain independent representations in bilingual settings. In our work we try to decouple language and domain specific representations through adapter layers.

Adapter layers Bapna and Firat (2019) introduce adapter layers for NMT as a lightweight alternative to fine-tuning. They study both adding language-pair specific adapters to multilingual NMT models to match the performance of a bilingual version, and domain-specific adapters for parameter-efficient domain adaptation. Philip et al. (2020) train adapters for each *language* instead of *language-pair* and show that composing such adapters improves zero-shot translation in English-centric settings, and can adapt a model to all language directions in a scalable way. Pfeiffer et al. (2020, 2021) study adapter layers for pre-trained language models evaluated on NLU tasks. They show it is possible to compose language and task adapters. Combining language adapters trained with a masked language modelling objective for language x and task adapters trained on a classification task in language y can transfer to classification in language x. We have a similar objective to Pfeiffer et al. (2020), but for NMT where in addition to encoding sentences we need to generate text for new language and domain combinations.

To the best of our knowledge none of the works above study composing language and domain adapters for generation tasks (such as translation) which is the goal of this work.

3 Composing Adapter Modules

Adapter modules (Rebuffi et al., 2017; Houlsby et al., 2019) are randomly initialised modules inserted between the layers of a pre-trained network and fine-tuned on new data. An adapter layer is typically a down projection to a bottleneck dimension followed by an up projection to the initial dimension, which we write as $\text{FFN}(\mathbf{h}) = W_{\text{up}} f(W_{\text{down}}\mathbf{h})$, with $f(\cdot)$ a non-linearity. The bottleneck controls the parameter count of the module; typically NMT requires slightly larger parameter counts than classification to match fine-tuning (Bapna and Firat, 2019; Cooper Stickland et al., 2021). With a residual connection and a near-identity initialization the original model is (approximately) retained at the beginning of optimization, keeping at least the performance of the parent model.

3.1 Stacking Domain and Language Adapters

In this work we study 'stacking' adapter modules, i.e. each language and domain has a unique adapter module associated with it. When passing a batch with source language x, target language y, and domain z, we only 'activate' the adapters for $\{x, y, z\}$. The encoder adapters for x and decoder adapters for y are activated.

We mostly follow the architecture of Bapna and Firat (2019). Language adapters LA are defined as:

$$\text{LA}(\mathbf{h}_l) = \text{FFN}_{\text{lg}}(\text{LN}_{\text{lg}}(\mathbf{h}_l)) + \mathbf{h}_l \quad (1)$$

where $\mathbf{h}_l$ is the Transformer hidden state at layer l and LN_{lg} is a newly initialised layer-norm. Let $\mathbf{z} = \text{LA}(\mathbf{h}_l)$; when stacking domain and language adapters, the layer output $\mathbf{h}_{l,\text{out}}$ is given by:

$$\mathbf{h}_{l,\text{out}} = \text{FFN}_{\text{dom}}(\text{LN}_{\text{dom}}(\mathbf{z})) + \mathbf{z} \quad (2)$$

For all models without any stacking we obtain layer output as in Eq. 2 but replace $\text{LA}(\cdot)$ with the identity operation.

Pfeiffer et al. (2020) use a different formulation that empirically performed well for them, but that in initial experiments produced worse results in our setting. We list the corresponding equations and results in Appendix B and Appendix D.

3.2 Improving the Compositionality of Adapters

In our initial experiments (Section 5.2) we found that (unlike Pfeiffer et al., 2020) naive stacking of language and domain adapters does not work very well for unseen combinations of language and domains, and often results in off-target translation (i.e. translations into the wrong language). Therefore, we study several strategies to improve the compositionality of adapters in the context of NMT:

1) Using **decoder-only** domain adapters when translating from an out-of-domain source language into an in-domain[1] target language, and **encoder-only** domain adapters when translating from an in-domain source language into an out-of-domain target language. This means we never stack together a combination of language and domain adapter that was not seen at training time. We also find empirically that decoder-only adapters work well with back-translation, perhaps because they can 'ignore' the noisy synthetic source-side data.

2) **Domain adapter dropout** (DADrop). Similar to layer-drop (Fan et al., 2020b) but specialised to adapter layers, or AdapterDrop (Rücklé et al., 2020) but without targeting specific layers, we randomly 'drop' (i.e. skip) the domain adapter[2] and only pass the hidden state through the language adapter. This means the adapter stack in the layer above can more easily adapt to unfamiliar input, and encourages domain and language adapters to be more independent of each other.

3) **Data augmentation**. We often have access to monolingual data in a domain even when no parallel data is available. In this work we leverage English-centric back-translation (BT), i.e. translating monolingual data in some languages into English (thus avoiding the more expensive step of translating from each language into every other language). We examine the ability of such data to help cross-lingual transfer to unseen combinations of source and target language (BT means we have artificial data for every language in combination with English). We briefly explore '*denoising auto-encoder*' style objectives as in unsupervised MT (Lample et al., 2018) or sequence-to-sequence pre-training (Lewis et al., 2020).

[1] Reminder we refer to the subset of languages we have parallel data for in a particular domain as 'in-domain', and all other languages as 'out-of-domain'.

[2] We could additionally drop the language adapter, but since this was frozen in many experiments we limit ourselves to domain adapters for simplicity

4 Experimental Settings

4.1 Data

For studying the domain transfer across languages we select four diverse domains that have data available in most language directions: translations of the Koran (**Koran**); medical text from the European Medicines Agency (**Medical**); translation of TED Talks transcriptions (**TED**); various technical IT text, e.g. the Ubuntu manual (**IT**). All data was obtained from the OPUS repository (Tiedemann, 2012). We create validation and test sets of around 2000 sentences each, and avoid overlap with training data (including parallel sentences in any language) with a procedure described in Appendix A. Note that Medical, Koran and IT are from the same source as those of Aharoni and Goldberg (2020), although the train/test splits are different due to expanding the number of languages and wanting a consistent pipeline for obtaining the data.

Domain	Langs.	Avg size (lines)
ParaCrawl	12	125M
Koran	10†	52k
Medical	11‡	500k
IT	12	196k
TED	12	138k

Table 1: Basic statistics for the datasets we use; number of languages covered, and average number of training examples across all language directions. †: missing nb & da, ‡: missing nb.

4.2 Models

In **multilingual settings** we concentrate on 12 high-resource European languages[3] due to the availability of domain-specific parallel data for most language pairs. Our **baseline model** is a Transformer Base (Vaswani et al., 2017) trained on English-centric ParaCrawl v7.1 data (Bañón et al., 2020) with all 12 languages (803M line pairs in total). It is trained with fairseq (Ott et al., 2019) for 800k updates, with a batch size of maximum 4000 tokens and accumulated gradients over 64 steps (Ott et al., 2018).[4] The source/target embeddings are shared and tied with the output layer. We tokenize the data with a shared BPE model of size 64k with inline casing (Berard et al., 2019b) Both the multilingual models and BPE model are trained with temperature-based sampling with $T = 5$ (Arivazhagan et al., 2019). We calculate all BLEU scores with Sacrebleu[5] (Post, 2018). On the recommendation of Marie et al. (2021) we additionally report chrF (Popović, 2015) calculated using Sacrebleu[6] for most models in the Appendix. We use adapter bottleneck size of 1024 unless stated otherwise, and when using DADrop (Section 3.2) use a 20% chance of skipping the domain adapter.

We additionally train monolingual language adapters (Philip et al., 2020) for all 12 languages on multi-parallel ParaCrawl data, which we obtain by aligning all languages through their English side, like Freitag and Firat (2020). The adapters are trained for another 1M steps, without accumulated gradients. We report the results of models fine-tuned on both all the domains simultaneously, or each domain separately, with access to in-domain data available for all the languages. Both serve as a potential upper bound for cross-lingual transfer.

We train the same model (i.e. with access to all languages) with domain tags: one special token per domain prepended to each source sequence (Kobus et al., 2017). We also measure the cross-lingual transfer ability of domain tags, by training a model with domain tags on all 4 domains but with in-domain data in only 4 languages (fr, de, cs and en). Because the latter model exhibits catastrophic forgetting issues in the other languages, we also train the same model with ParaCrawl data in all language directions (with a "paracrawl" domain tag). ParaCrawl line pairs are sampled with probability 0.5. More training hyper-parameters are given in Appendix A.

4.3 Our model pipelines

We perform two series of experiments.

Multilingual multi-domain models. Firstly, we experiment with different ways of multi-domain adaptation of multilingual models. We adapt the English-centric ParaCrawl pre-trained model to four domains (Koran, Medical, IT and TED) and every language direction simultaneously. We test models with language adapters, language + domain adapters, and domain tags. There is no cross-lingual domain transfer needed[7] since all language

[3] {cs, da, de, en, es, fr, it, nb, nl, pl, pt, sv}

[4] This corresponds to an effective batch size of ≈207k tokens and training length of 7 epochs.

[5] Signature: BLEU+case.mixed+lang.m2m-en+numrefs.1+smooth.exp+tok.13a+version.1.5.0.

[6] Signature: chrF2+numchars.6+space.false+version.1.5.1

[7] There is obviously cross-lingual domain transfer that may take place when all the domains are trained jointly, but we do

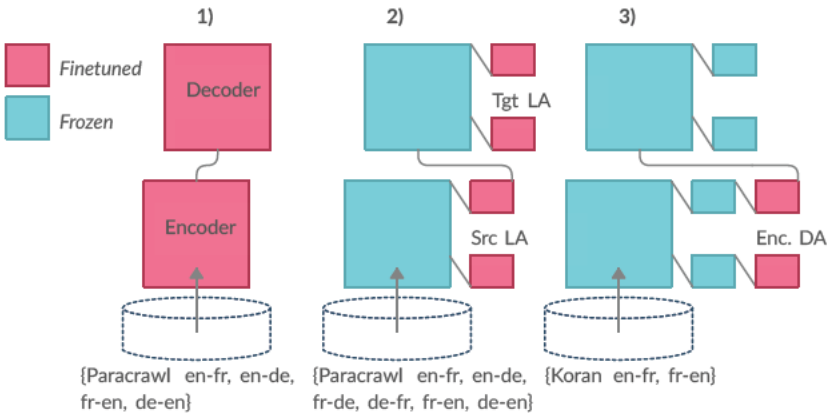


Figure 1: Toy diagram showing one of our proposed pipelines for training language and domain adapters, on a example subset of languages: {en,fr,de}, with 'domain-agnostic' data from ParaCrawl and specialised data from the Koran. Red indicates a fine-tuned model component, blue indicates a frozen component. LA = language adapter, DA = domain adapter. From left to right we show: 1) Training an encoder-decoder model with English-centric ParaCrawl. 2) Training mononlingual language adapters with multiparallel Paracrawl data. 3) Training domain adapters stacked on language adapters in the encoder, on a subset (here {en, fr}) of languages for the domain of interest (e.g. Koran). Here we show domain adapters added only to the encoder, but we consider various other configurations in this work.

directions are included in the training data. Results for this scenario are reported in Section 5.1.

Cross-lingual domain transfer. In the second experiment we try to decouple the notion of domain from language via analysing the zero-shot composition of domain and language adapters. This is described in a toy diagram in Figure 1. We first extend the baseline multilingual English-centric model with 12 (one for each language) monolingual language adapters (Philip et al., 2020) trained on multi-parallel ParaCrawl. We then test the cross-lingual domain transfer ability of our proposed combinations of adapters by training on data in a particular domain with a subset of four languages (referred to as '**in-domain**'; in Figure 1 *en* and *fr* would be in-domain). We test our model on all language directions from the set of all twelve languages. This will include cases where we don't have in-domain data for either the source or target language, which we refer to as '**out-of-domain**' (in Figure 1 *de* would be out-of-domain).

Finally, we extend the above mentioned scenario with back-translated (BT) data from *out-of-domain* languages into English. To create the BT data, we use the model with language adapters trained on ParaCrawl (**11**) (which has not seen any in-domain data) on the English-aligned training data for each language and domain, and use beam search with a beam size of 5. Results for this scenario are reported in Section 5.2.

To train language and domain adapters, we freeze all model parameters except for adapter parameters, and use a fixed learning rate schedule with learning rate 5×10^{-5}. Following Philip et al. (2020), when training language adapters without domain adapters we build homogeneous batches (i.e. only containing sentences for one language direction) and activate only the corresponding adapters. When training language *and* domain adapters together, we build homogeneous batches that only contain sentences for the same combination of language direction and domain.

5 Results and Discussion

First, in Section 5.1 we discuss the results of experiments testing the domain adaptation capacity of various models, assuming access to data for all language pairs. In Section 5.2 we analyse domain transfer across languages with adapters and other methods. We first demonstrate problems with cross-lingual generalisation during domain adaption for 'naive' methods, and then propose potential solutions. Note we concentrate on the medical domain and a particular language subset for convenience. Appendix D has results in

not explicitly study this in the first experiment.

ID	Model	IT	Koran	Medical	TED	Params (M)
(1)	Base (En-centric)	23.2	7.0	25.7	19.0	N/A
(2)	Finetuned	40.8	16.0	42.7	26.6	79
(3)	Finetuned + domain tags	**43.6**	**20.3**	**46.0**	27.2	79
(4)	Single adapter per layer ($d = 1024$)	39.6	14.7	41.8	26.2	12.6
(5)	LA ($d = 1365$)	42.0	17.6	43.7	26.8	202
(6)	LA ($d = 2048$)	42.2	18.1	43.8	26.9	303
(7)	LA + dec. DA ($d = 1024$)	42.1	18.5	43.6	27.1	177
(8)	LA + enc. DA ($d = 1024$)	42.3	19.3	43.8	27.5	177
(9)	LA + enc & dec. DA ($d = 1024$)	42.7	20.1	44.0	**27.7**	202

Table 2: BLEU scores averaged across all the language-directions for various multilingual multi-domain adaptation strategies, i.e. training on all language directions from the 12 languages and all domains. LA = language adapters, DA = domain adapters. 'Params (M)' refers to the number of trainable parameters in millions. Note that unlike in Table 3 the LA here are not pre-trained on ParaCrawl; they are trained jointly with domain adapters.

ID	Model	All	In→in	Out→in	In→out	Out→out
	Oracles					
(10)	Finetune (all langs)	44.3	43.9	44.8	44.2	44.2
(3)	FT (all langs & domains) + dom. tags	46.0	45.3	46.3	45.9	46.0
	Baselines					
(1)	Base (En-centric)	25.7	27.0	27.2	25.9	24.3
(11)	**(1)** + ParaCrawl LA	30.2	29.6	30.8	30.0	30.0
	Straightforward Methods					
(12)	**(1)** + Domain adapters only	23.0	44.7	37.8	13.4 (7%)	13.4 (11%)
(13)	Freeze LA + enc. & dec. DA	26.9	44.0	36.7	20.1 (71%)	19.9 (76%)
(14)	Freeze LA + enc. DA	29.6	42.6	34.0	27.0 (89%)	24.6 (88%)
(15)	Freeze LA + dec. DA	29.0	41.7	**40.7**	22.5 (77%)	22.0 (77%)
(16)	FT (all domains) + dom. tags	15.6	**46.8**	13.2 (55%)	12.0 (1%)	10.7 (2%)
	Improving Off-target Translation					
(17)	**(16)** + ParaCrawl	34.7	42.2	39.6	32.4	31.0
(18)	**(13)** + BT	33.9	43.2	36.8	35.9	28.0 (85%)
(19)	**(14)** + BT	32.5	41.8	35.0	34.7	26.8 (83%)
(20)	**(15)** + BT	**36.9**	40.9	38.2	36.4	**35.1**
(21)	**(13)** + DADrop	28.0	42.6	36.7	22.9 (82%)	21.5 (82%)
(22)	**(13)** + BT + DADrop	34.8	42.2	37.0	36.5	30.2
(23)	Unfreeze LA + dec. DA	14.3	45.8	36.6	0.0 (1%)	0.0 (2%)
(24)	**(23)** + DADrop	31.1	45.5	36.9	23.9 (82%)	27.8
(25)	**(23)** + DADrop + BT	35.2	44.5	33.4	**38.2**	31.8

Table 3: BLEU score of various models trained on the {en, fr, de, cs} subset of the Medical domain, except 'Oracle' models trained on all language pairs. LA = language adapters, DA = domain adapters. 'Out→in' is the average score when translating from an out-of-domain source language into {en, fr, de, cs}. 'In→out' corresponds to when the out-of-domain language is the target language. 'In→in' refers to average score when source and target are in the set {en, fr, de, cs}. 'Out→out' is the average score when both the source and target language are unseen during domain adaptation. We note percentage of **on-target** (correct language) translations in brackets, when it is less than 90% only.

other domains and language subsets, and also chrF (Popović, 2015) scores; we find similar trends to those reported in Section 5.2.

5.1 Multilingual multi-domain models

Table 2 reports the results from the challenging task of adapting a multilingual NMT model to multiple domains and language directions simultaneously. In this scenario, we assume access to in-domain data in all the language directions, and so we are testing the capacity of various models for domain adaptation, rather than cross-lingual transfer. Models are compared against a baseline **(1)** not trained on in-domain data.

We report the results for naive fine-tuning on the concatenation of in-domain parallel datasets for all the languages and all the domains **(2)**. On all domains we improve on these results by fine-tuning with domain tags **(3)** (a similar result to Jiang et al. (2020) in the bilingual setting).

Fine-tuning with domain tags **(3)** outperforms the model with stacked adapters **(9)**. A fine-grained comparison of these models is in Figure 5 in the Appendix. For the IT and Medical domains the model with tags **(3)** is clearly better for all language directions. For the lowest resource domains, Koran and TED, most of the differences are not statistically significant, except for English-centric language pairs for TED, where the adapter model **(9)** is better. Exploring the combination of domain tags and adapters could be an interesting future research direction.

Stacking domain and language adapters **(9)** results in better performance than a model with the same parameter budget devoted to language adapters only **(5)**. We believe this is because it allows the model to (partially) decouple domain from language-specific information, and better exploit the allocated parameter budget. Even a higher capacity language adapter model **(6)** does not perform as well.

We also note that usage of encoder-only domain adapters **(8)** outperforms the decoder-only domain adapter model **(7)**. This is perhaps because the encoder representations influence the whole model (it is directly connected to the decoder at all layers with encoder-decoder attention) as opposed to the the decoder adapters that only impact decoder representations. We find a similar trend in bilingual domain adaptation, see Appendix C.

The strong performance of encoder-only adapters has interesting implications for inference speed. With an auto-regressive decoder, the computational bottleneck is on the decoder side. The encoder output is computed all at once, while computing the decoder output requires L steps, where L is the output length. This implies devoting more capacity to encoder adapters would achieve similar performance and faster inference (more details in Appendix C).

5.2 Cross-lingual Domain Transfer

To study the capacity of our models to transfer domain knowledge across languages, we perform domain adaptation using parallel datasets for a *subset* of language pairs, and evaluate on the test sets available for *all* language pairs. In this section we report the results for adaptation to the *medical* domain using the subset of all the language directions including {en, fr, de, cs} languages (Table 3). We refer to these languages as *in-domain* languages, and *out-of-domain* languages would include all the other languages, {de, nl, sv, es, it, pt, pl } (referred to as *In* and *Out* respectively in Table 3).

We report BLEU scores averaged across test sets of different categories of language-directions depending on whether the source/target language was observed during the domain adaptation training: In→in for language pairs observed during DA, Out→out for fully zero-shot DA performance, and In→out, Out→in for translation directions combining *in-domain* and *out-of-domain* languages.

First, we report the results for *Oracle* models providing an upper bound for the scores models could achieve with access to in-domain data for *all* the languages: model **(10)** was fine-tuned on *medical* data for all the language directions[8], and a model with domain tags **(3)** discussed in section 5.1.

Baseline models include the default multilingual English-centric model **(1)**, as well as model **(11)** with language adapters trained on multi-parallel ParaCrawl data. Comparing against this baseline shows us improvements from domain-specific (rather than language-specific) information.

Straightforward Methods We train several 'straightforward' adapter models for the subset of *in-domain* languages on the top of the baseline model, one with no language adapters, model **(12)**, and model **(13)** with domain *and* language adapters

[8]This is different from the model **(3)** which was fine-tuned on *all* the domains and all the language directions.

(where language adapters are frozen), stacking them in the encoder and decoder.

Both of these models achieve good scores when translating into *in-domain* languages (the In→in and Out→in categories), on par or better than *Oracle* scores and much higher than the baselines. On the other hand they suffer from significant drops in performance when translating into *out-of-domain* languages (the In→out and Out→out categories).

The model **(16)** trained with tags on a subset of *in-domain* languages suffers from the same low performance translating into *out-of-domain* languages and additionally has low performance with out-of-domain *source* languages.

Looking closer at the translations of the above models, we see that many translations are either generated in English, copy the source language, or mix words between English and the true target language; see Table 4 in the Appendix for illustrative examples. We refer to this phenomenon as "off-target" translation. We report the percentage of translations generated in the correct target language in Table 3 when it is lower than 90% [9].

We believe this phenomenon is partly due to decoder domain adapters having never been exposed to *out-of-domain* language generation. Encoder domain adapters seem to be less sensitive to composition with new language adapters (as observed by Pfeiffer et al. (2020) for NLU tasks, and Table 3 in the Out→In column).

To investigate this, we train models **(14)** and **(15)** with encoder-only and decoder-only adapters. Figure 2 compares the performances of these models as well as model **(13)** trained with encoder and decoder domain adapters, **(14)**, **(15)** against the baseline model **(11)**. The decoder-only model **(15)** can better translate *from* out-of-domain languages and the encoder-only model **(14)** slightly improves for translations *into* out-of-domain languages. However the problem of off-target translation persists for both models and neither improves over ParaCrawl LA **(11)**. Therefore, we conclude that a straightforward combination of domain and language adapters leads to catastrophic forgetting both in the encoder and the decoder, but the encoder is less important for this effect.

[9]This percentage is computed against the reference translations that were correctly tagged by 'langdetect', a Python language identifier (`https://pypi.org/project/langdetect/`). This is to exclude very short and numerical examples which can be quite frequent in some domains.

Effect of data augmentation We train models **(17)**,**(18)** **(19)**, **(20)** with additional data (either a portion of ParaCrawl data, or back-translation of in-domain data) to alleviate potential forgetting of representations for *out-of-domain* languages. All of these models improve the translation quality into *out-of-domain* languages. The model with tags **(17)** reaches competitive results and can be considered as a strong baseline.

For models with back-translation data, the decoder-only adapter **(20)** model outperforms the encoder-only adapter **(19)** model on out-of-domain target languages (as opposed to the case without BT) and has the strongest results overall on translating into out-of-domain languages. While the BT models are trained on exactly the same data, this effect is possibly due the encoder adapters being more influenced by potentially noisy synthetic source-side data, whereas decoder adapters are more influenced by clean reference translations. The decoder-only BT model **(20)** improves over the baseline for all the language directions except for translation into English; see Figure 3.

We report results for the other data augmentation methods (see Section 3.2) in Appendix D; these only improve over the ParaCrawl LA baseline in limited settings.

Domain adapter dropout Models **(21)** and **(22)** trained with dropping domain adapters (DADrop; see Section 3.2) also allow to reduce catastrophic forgetting, although only combining DADrop with Data Augmentation (model **(22)**) allows to solve the problem of off-target translation. We also note slight decreases in *in-domain* performance for those models, perhaps due to underfitting.

Increasing adaptation capacity When naively increasing model capacity by unfreezing LA stacked with decoder DA **(23)**, the model seems to mostly devote this capacity to In→in category, and suffers on other language pair groups. This trend seems to be similar to the un-augmented model with tags **(16)**. However, once regularized with DADrop **(24)**, and augmented with back-translation **(25)** it reaches very competitive results.

Figure 4 shows fine-grained results for different models with DADrop, back-translation and unfrozen LA. Back-translation improves performance on the Out→out and In→out groups, but decreases performance on the Out→in group. Finally, unfreezing language adapters decreases the perfor-

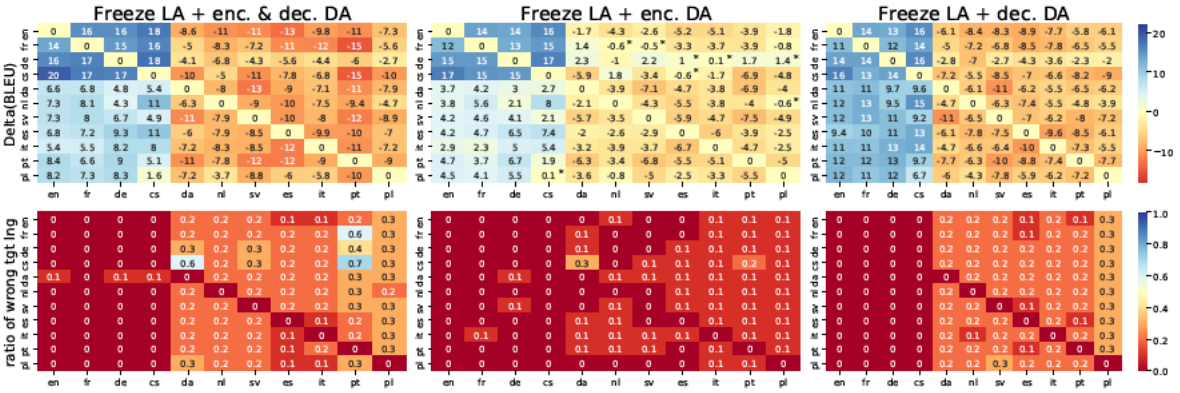


Figure 2: Comparing models with encoder-decoder adapters, encoder-only adapters and decoder-only adapters. x-axis shows the target language and y-axis shows the source language. Languages are grouped so the in-domain languages are in the top left corner. Top: Difference in BLEU compared to the baseline (**11**) (negative scores indicate a decrease w.r.t. the baseline, "*" indicates *not* statistically significant). Bottom: proportion translating into the wrong target language. Best viewed in .pdf form.

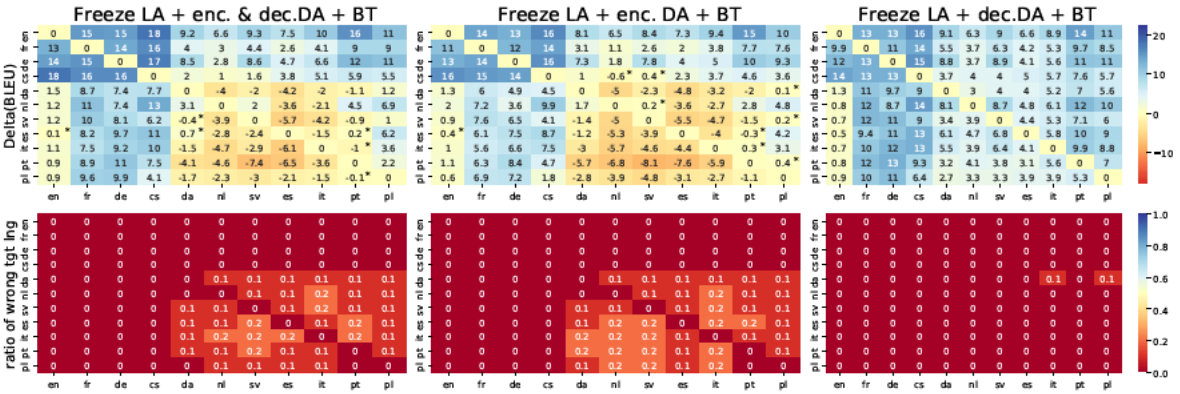


Figure 3: Comparing adapter models trained with back-translation. Top: Difference in BLEU compared to the baseline (**11**) ("*" indicates *not* statistically significant). Bottom: proportion translating into the wrong target language. See Figure 2 for more details.

mance on Out→in but improves on the Out→out group.

Adapters vs. tags As mentioned previously, model (**17**) with tags augmenteed with ParaCrawl reaches competitive scores overall. Note that this model was trained on a concatenation of *all* the domains, unlike the models with adapters which were trained *only* on the medical domain. Therefore it has been exposed to more data overall. On the other hand, several of our models fine-tune only a single adapter per-layer and use frozen LA. Thus, encoder-only or decoder-only models only require 6.3 million tunable parameters, compared to 79 million for tag-based models. Additionally adapter models can easily be 'mixed-and-matched' by activating a particular adapter for a particular language pair. For example we could activate model (**15**) on 'Out→in' (out-of-domain source, in-domain target) data, model (**18**) on in-domain data and model (**20**) otherwise. Such models could easily be extended to new domains by training more adapters, in contrast to tag-based models which update all parameters for each domain adaptation request.

6 Conclusion

In this work we studied multilingual domain adaptation both in the full resource setting where in-domain parallel data is available for all the language pairs, as well as the partial resource setting, where in-domain data is only available for a small set of languages.

In particular, we study how to better compose language and domain adapter modules in the context of NMT. We find that while adapters for encoder architectures like BERT can be safely com-

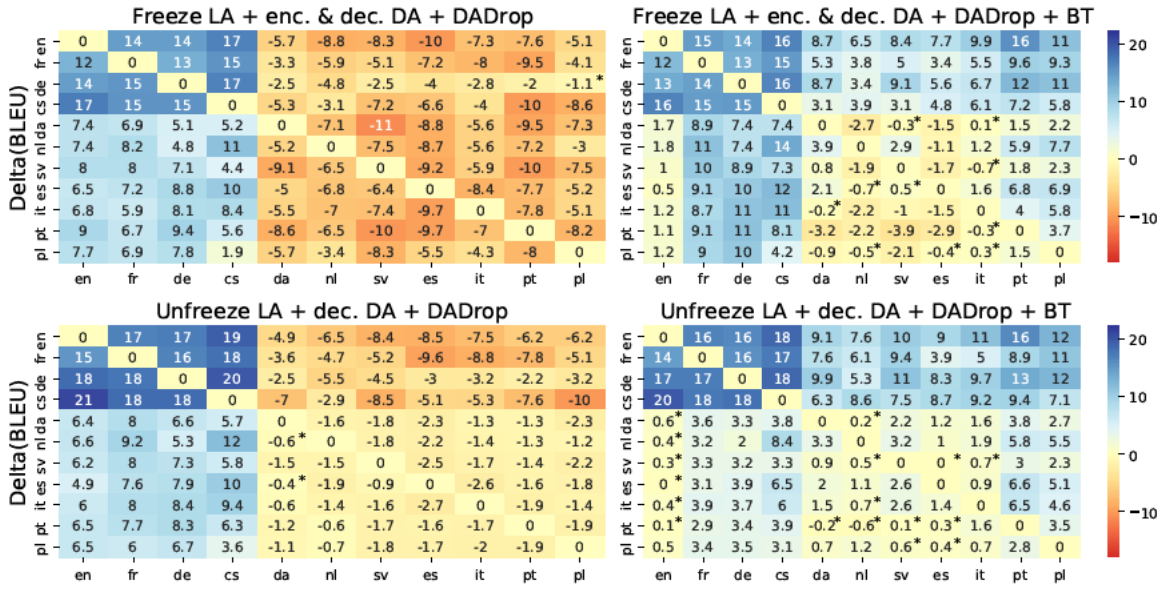


Figure 4: Comparing models with DADrop, back-translation and unfrozen language adapters. Difference in BLEU compared to the baseline (**11**) ("*" indicates *not* statistically significant). See Figure 2 for more details.

posed, this is not true for NMT adapters: domain adapters learnt in the partial resource scenario struggle to generate into languages they were not trained on, even though the original model they are inserted in was trained on those languages. We found that randomly dropping the domain adapter and back-translation can regularize the training and lead to less catastrophic forgetting for when generating into out-of-domain languages, although they do not fully solve the problem of off-target translation.

We experimented with different adapter placement and found that devoting additional capacity to encoder adapters can lead to better results compared to when the same capacity is shared between the encoder and the decoder. Similarly, in the partial resource scenario, models with encoder-only domain adapters suffer less from catastrophic forgetting when translating into out-of-domain languages. In contrast, decoder-only domain adapters perform well when translating *from* out-of-domain into in-domain languages, and combine well with back-translation, perhaps due to their ability to ignore noisy synthetic source data.

Finally we note that a model fine-tuned with domain tags serves as a very competitive baseline for multilingual domain adaptation. On the other hand, domain adaptation with adapters offers modularity, and allows incrementally adapting to new domains without retraining the full model. Future research directions could explore multi-task training combining parallel and monolingual in-domain data in other ways to alleviate the need for back-translation.

Our work is the first attempt to combine domain adapters and language adapters for a generation task (NMT). Although such combinations have shown to be successful for NLU tasks, obtaining good representations for generating unseen target languages proves to be a difficult problem. We believe a fine-grained study of where to use language or domain-specific capacity could lead to better cross-lingual domain transfer in future. Finally, we provide supplementary material to facilitate reproducibility.[10]

Acknowledgements

We would like to thank Laurent Besacier, Hady Elsahar, Matthias Gallé, Germán Kruszewski, Ahmet Ustun and all of the NAVER Labs Europe team for useful discussions. Asa Cooper Stickland was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

References

Roee Aharoni and Yoav Goldberg. 2020. Unsupervised domain clusters in pretrained language models. In

[10] https://tinyurl.com/r66stbxj

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics.

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. Association for Computational Linguistics.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George F. Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *CoRR*, abs/1907.05019.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. 2020. ParaCrawl: Web-scale acquisition of parallel corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, Online. Association for Computational Linguistics.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Alexandre Berard, Ioan Calapodescu, Marc Dymetman, Claude Roux, Jean-Luc Meunier, and Vassilina Nikoulina. 2019a. Machine translation of restaurant reviews: New corpus for domain adaptation and robustness. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 168–176, Hong Kong. Association for Computational Linguistics.

Alexandre Berard, Ioan Calapodescu, and Claude Roux. 2019b. Naver Labs Europe's systems for the WMT19 machine translation robustness task. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 526–532, Florence, Italy. Association for Computational Linguistics.

Denny Britz, Quoc Le, and Reid Pryzant. 2017. Effective domain mixing for neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 118–126, Copenhagen, Denmark. Association for Computational Linguistics.

Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020. TyDi QA: A Benchmark for Information-Seeking Question Answering in Typologically Diverse Languages. In *Transactions of the Association of Computational Linguistics*.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. 2018. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of EMNLP 2018*, pages 2475–2485.

Asa Cooper Stickland, Xian Li, and Marjan Ghazvininejad. 2021. Recipes for adapting pre-trained monolingual and multilingual models to machine translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3440–3453, Online. Association for Computational Linguistics.

Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. 2020. A comprehensive survey of multilingual neural machine translation. *arXiv preprint arXiv:2001.01115*.

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

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Edouard Grave, Michael Auli, and Armand Joulin. 2020a. Beyond english-centric multilingual machine translation.

Angela Fan, Edouard Grave, and Armand Joulin. 2020b. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

Markus Freitag and Yaser Al-Onaizan. 2016. Fast domain adaptation for neural machine translation. *arXiv preprint arXiv:1612.06897*.

Markus Freitag and Orhan Firat. 2020. Complete multilingual neural machine translation. In *Proceedings of the Fifth Conference on Machine Translation*, pages 550–560, Online. Association for Computational Linguistics.

Xavier Garcia, Noah Constant, Ankur P. Parikh, and Orhan Firat. 2021. Towards continual learning for multilingual machine translation via vocabulary substitution. *CoRR*, abs/2103.06799.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, volume 97, pages 2790–2799, Long Beach, California, USA.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. *CoRR*, abs/2003.11080.

Haoming Jiang, Chen Liang, Chong Wang, and Tuo Zhao. 2020. Multi-domain neural machine translation with word-level adaptive layer-wise domain mixing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1823–1834, Online. Association for Computational Linguistics.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. A simple baseline to semi-supervised domain adaptation for machine translation.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Karthikeyan K, Zihan Wang, Stephen Mayhew, and Dan Roth. 2020. Cross-lingual ability of multilingual bert: An empirical study. In *International Conference on Learning Representations*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Catherine Kobus, Josep Crego, and Jean Senellart. 2017. Domain Control for Neural Machine Translation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 372–378, Varna, Bulgaria. INCOMA Ltd.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations (ICLR)*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domain. In *International Workshop on Spoken Language Translation*.

Benjamin Marie, Atsushi Fujita, and Raphael Rubino. 2021. Scientific credibility of machine translation research: A meta-evaluation of 769 papers. *CoRR*, abs/2106.15195.

M. McCloskey and N. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165.

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 875–880, Brussels, Belgium. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling Neural Machine Translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th*

Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jerin Philip, Alexandre Berard, Matthias Gallé, and Laurent Besacier. 2020. Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online. Association for Computational Linguistics.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, Long Beach, California, USA.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *CoRR*, abs/2010.11918.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Emmanouil Stergiadis, Satendra Kumar, Fedor Kovalev, and Pavel Levin. 2021. Multi-domain adaptation in neural machine translation through multidimensional tagging.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, Long Beach, California, USA.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1628–1639, Online. Association for Computational Linguistics.

A Data and Hyper-parameters

For **bilingual** domain adaptation we use a Transformer Base (Vaswani et al., 2017) model trained for 12 epochs on German to English WMT20 data (47M parallel lines), with a joint BPE (Sennrich et al., 2016b) vocabulary of size 24k with inline casing (Berard et al., 2019b) (i.e. wordpieces are put in lowercase with a special token indicating their case.). For bilingual domain adaption we use the same datasets as Aharoni and Goldberg (2020), namely parallel text in German and English from five diverse domains: Koran, Medical, IT, Law and Subtitles.

For multilingual settings we use the following hyper-parameters. We share embeddings between encoder and decoder. We use the Adam optimizer (Kingma and Ba, 2014) with an inverse square root learning rate schedule for pre-training, and a fixed learning rate schedule for training adapters. We speed up training with 16 bit floating point arithmetic. We use label smoothing 0.1 and dropout 0.1. We train for either 20 epochs or 1 million updates, whichever corresponds to the smallest number of training updates. We use early stopping, checking performance after each epoch or every 100,000 training steps, and use average validation negative-log-likelihood on all of the training data (but not out-of-domain language data) as our criteria for choosing the best model. We otherwise use default Fairseq (Ott et al., 2019) parameters. We train all models on a single Nvidia V100 GPU, and training takes between 8 and 36 hours depending on dataset size.

In order to create validation and test splits that had no overlap with training data in any language, we first set aside a number of English sentences. Then we aligned all language pairs to these sentences, i.e. the German to French test set is composed of German and French sentences that share the same English sentence. Finally we remove all sentences in any language from the train splits of all parallel data if those sentences are aligned with any English sentences in the subset we set aside for validation/test splits. Both validation sets and test sets contain around 2000 examples for every domain and language-pair.

B MAD-X Style Stacking

Pfeiffer et al. (2020) use the following stacking formulation,

$$\mathrm{LA}(\mathbf{h}_l, \mathbf{r}_l) = \mathrm{FFN}_{\mathrm{lg}}(\mathbf{h}_l) + \mathbf{r}_l. \quad (3)$$

The residual connection $\mathbf{r}_l$ is the output of the Transformer's feed-forward layer whereas $\mathbf{h}_l$ is the output of the subsequent layer normalisation. When stacking domain and language adapters the layer output is given by applying the model's pre-trained layer norm $\mathrm{LN}_{\mathrm{pre}}$,

$$\mathbf{h}_{l,\mathrm{out}} = \mathrm{LN}_{\mathrm{pre}}(\mathrm{FFN}_{\mathrm{dom}}(\mathrm{LA}(\mathbf{h}_l, \mathbf{r}_l)) + \mathbf{r}_l) \quad (4)$$

and using the output of the Transformer's feed-forward layer as a residual instead of the language adapter output. We refer to this as '**MAD-X**' style after Pfeiffer et al. (2020). This leaves the layer output 'closer' to the pre-trained model, with the same layer-norm and residual connection, contrary to Eq. 2 which has a newly initialised layer-norm and a residual connection. For all models without any stacking we obtain layer output as in Eq. 4 but replace $\mathrm{LA}(\cdot)$ with the identity operation.

C Additional Results for Bilingual Domain Adaptation

Before studying multilingual domain adaptation, we validate some of our ideas on a simpler, *bilingual* German → English domain adaptation setting. Table 11 reports the results of this experiment. First, we note that *encoder-only* adapters perform similarly to *encoder & decoder* adapters, while *decoder-only* adapters perform worse.

Moreover, adding adapters to only the last three layers of the encoder almost matches the performance of adapting every layer, while adding adapters to the first three layers decreases performance. We believe this is because the last encoder layer directly influences every layer of the decoder through cross-attention.

Table 12 presents results of bilingual domain adaption with smaller adapter bottleneck dimension. The same trends emerge: encoder-only adapters perform better, and the last three layers of the encoder are better than the first three. The last three encoder layers also perform better than the first three for a multilingual model, see Table 7 models **(38)** and **(39)**. Interestingly the multilingual last three encoder layer DA model is roughly halfway between encoder-only and decoder-only on Out→in and In→out performance, suggesting it might be a useful compromise between the two.

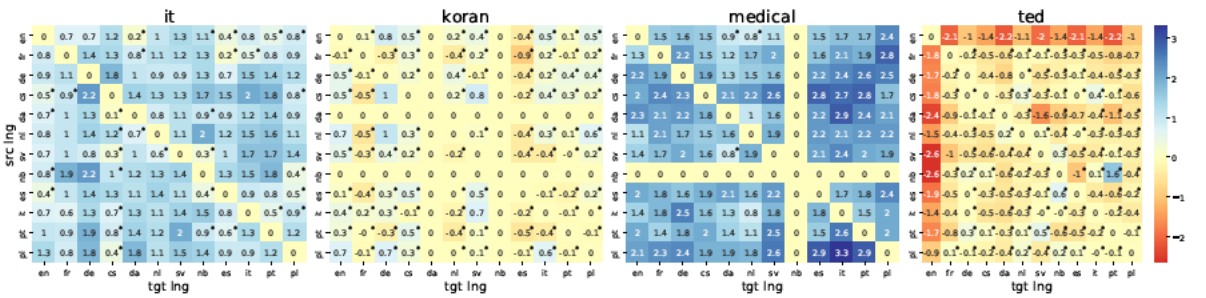


Figure 5: Difference in BLEU score for each domain between the model trained with adapters (9) and model trained with domain tags (3), for the multilingual multi-domain models. A positive number corresponds to the case where model (3) has higher score than the model (9). The "*" indicates the cases when the difference is *not* statistically significant.

D Additional Results for Cross-lingual Transfer

Does language diversity increase transfer? Table 5 compares models trained on a mix of language families (fr, de, cs, en) and mostly romance languages (fr, it, es, en) to test whether diversity of languages in our in-domain training set improved transfer. Positive numbers in this table indicate diversity of training languages improves performance. Diversity helps for translating out-of-domain languages into in-domain. We have unclear results for when both source and target are out-of-domain; it seems when using back-translation (BT), i.e. when all languages have been seen (albeit with artificial English parallel data) diversity helps, but without BT it mostly hurts performance. We speculate that training on mostly romance languages means the domain adapter encodes less 'language information', but leave further exploration to future work.

Additional results and metrics We present additional results for the setting discussed in Section 5.2 of the main paper in Table 9 (Koran domain), Table 10 (Koran results for the romance language subset), and Table 7 (additional Medical results). We use the chrF metric as discussed in the main paper, and find the conclusions based on BLEU score are unchanged. For the Koran domain, we see similar trends with decoder-only domain adapters (DA) performing best on out-of-domain source to in-domain target languages, and vice versa for encoder-only DA. Additionally we see as before that combining BT with decoder-only DA works the best, and achieves the highest overall performance. We report on-target (correct language) percentage for all medical domain models in Table 8.

We briefly experiment with denoising objectives, where we simply copy target data in out-of-domain languages to the source side (and optionally add 'noise' to the source side, e.g. swap tokens or mask tokens (Lewis et al., 2020)). Although we got reasonable improvements (models (42) and (41)) for out-of-domain target languages, we were mostly unable to improve over the pre-trained ParaCrawl LA, and so concentrate on back-translation.

We experiment with a setting where we jointly train on all language directions for IT, Koran and TED Talks domains and a subset of languages for Medical, and similarly with only a subset of Koran (models (43), (44) etc.). These models stack language and domain adapters. Such models don't require any pre-trained LA, and improve out-of-domain performance and decrease off-target translation compared to freezing ParaCrawl LA and training DA. However these scores are still worse than simply using pre-trained 'domain-agnostic' ParaCrawl LA (11).

source (fr)	ref (pt)	**(12)**	**(13)**
La durée du traitement dépend de la nature et de la sévérité de l' infection et de la réponse observée.	A duração do tratamento depende da natureza e da gravidade da infecção e da resposta verificada.	The duration of treatment depends on the nature and severity of the infection and on the response observed.	A duração do tratamento depende da nature e severidade da infecção e da resposta observed.
Insuman Comb 50 40 UI/ ml suspension injectable en flacon	Insuman Comb 50 40 UI/ ml, Suspensão injectável num frasco para injectáveis	Insuman Comb 50 40 IU/ ml suspension injectable en flacon	Insuman Comb 50 40 IU/ ml suspension for injection in vial
A quoi ressemble TAXOTERE et contenu de l' emballage extérieur TAXOTERE 80 mg, solution à diluer pour perfusion est une solution visqueuse, limpide, jaune à jaune marron.	Qual o aspecto de TAXOTERE e contéudo da embalagem TAXOTERE 80 - mg concentrado para solução para perfusão é uma solução viscosa transparente amarela ou amarela- acastanhada.	What TAXOTERE looks like and contents of the pack TAXOTERE 80 mg concentrate para solution for infusion is a solution visqueuse, limpida, de jaune à marron.	TAXOTERE 80 mg, Diluted for Solution for Infusion é uma solution viscous, limpa, yellow to marrom.

Table 4: Some examples of translations generated by straight-forward adapter training settings, in this case from a known source language, *fr* into a target language unseen during domain adaptation, *pt*, and for the medical domain.

Model	Out→ {en,fr}	Out→Out
Koran		
LA + Dec. DA	0.6	-0.9
LA + Dec. DA	0.3	-0.3
Unfr. LA + Dec. DA	0.1	-0.4
LA + Enc & Dec. DA	1.3	-1.3
Koran + BT		
LA + Dec. DA	0.4	0.2
LA + Dec. DA	1.9	0.9
Unfr. LA + Dec. DA	0.3	0
LA + Enc & Dec. DA	0.7	0.3
Medical		
LA + Dec. DA	0.8	-2.4
LA + Dec. DA	3.2	0.2
Unfr. LA + Dec. DA	0.2	0.1
LA + Enc & Dec. DA	3.2	-1.1
Medical + BT		
LA + Dec. DA	0.4	2.9
LA + Dec. DA	1.3	1.6
Unfr. LA + Dec. DA	0.5	3.1
LA + Enc & Dec. DA	0.5	1.5

Table 5: Difference in average BLEU score between models trained on a diverse subset of languages and models trained on mostly romance languages. Data source is noted in bold. Refer to the main paper for model definitions. Out→ {en,fr} corresponds to translation from an out-of-domain source language into {en,fr}. 'Out→Out' is the average score when both the source and target language are unseen during domain adaptation (choosing languages unseen by either subset).

ID	Model	IT	Koran	Medical	TED	Params (M)
(1)	Base (En-centric)	.456	.300	.488	.456	N/A
(2)	Finetuned	.623	.394	.625	.513	79
(3)	Finetuned + domain tags	**.645**	**.433**	**.646**	.517	79
(4)	Single adapter per layer ($d = 1024$)	.612	.382	.619	.512	12.6
(5)	LA ($d = 1365$)	.632	.408	.630	.517	202
(6)	LA ($d = 2048$)	.634	.411	.631	.517	303
(7)	LA + dec. DA ($d = 1024$)	.633	.412	.629	.518	177
(8)	LA + enc. DA ($d = 1024$)	.634	.426	.631	.522	177
(9)	LA + enc & dec. DA ($d = 1024$)	.636	.429	.632	**.523**	202

Table 6: chrF scores of various multilingual multi-domain adaptation strategies, i.e. training on all language directions from the 12 languages and all domains.

ID	Model	All	In→in	Out→in	In→out	Out→out
	Oracles					
(10)	Finetune (all langs)	.635	.631	.638	.635	.635
(3)	FT (all langs & domains) + domain tags	.646	.641	.648	.646	.646
	Baselines					
(1)	Base (En-centric)	.488	.500	.497	.493	.475
(11)	**(1)** + ParaCrawl LA	.537	.532	.540	.538	.535
	Straightforward Methods					
(12)	**(1)** + Domain adapters only	.400	.635	.575	.295	.287
(13)	Freeze LA + enc. & dec. DA	.468	.631	.566	.401	.400
(16)	FT (all domains) + dom. tags	.277	**.650**	.236	.283	.193
	Improving Off-target Translation					
(17)	**(16)** + ParaCrawl	.570	.622	.601	.556	.544
(26)	Unfreeze LA	.551	.639	.574	.514	.535
(27)	**(34)** + BT	.571	.636	.556	.594	.548
(15)	Freeze LA + dec. DA	.492	.613	**.605**	.432	.421
(20)	**(15)** + BT	**.586**	.608	.590	.584	**.577**
(28)	**(15)** + BT + DADrop	.584	.604	.587	.583	.576
(14)	Freeze LA + enc. DA	.518	.623	.548	.506	.477
(19)	**(14)** + BT	.548	.620	.567	.574	.498
(29)	**(14)** + BT + DADrop	.561	.614	.570	.579	.527
(30)	Freeze LA + enc. first 3 layers DA	.440	.618	.525	.379	.373
(31)	Freeze LA + enc. last 3 layers DA	.512	.622	.576	.472	.465
(21)	**(13)** + DADrop	.490	.621	.567	.447	.429
(32)	**(13)** + BT	.559	.626	.581	.582	.511
(22)	**(13)** + BT + DADrop	.569	.619	**.583**	.587	.534
(33)	**(13)** + BT + MAD-X style	.540	.625	.575	.561	.477
(23)	Unfreeze LA + dec. DA	.221	.641	.573	.010	.007
(24)	**(23)** + DADrop	.528	.639	.577	.452	.514
(25)	**(23)** + DADrop + BT	.573	.636	.559	.595	.550

Table 7: chrF score of various models trained on the {en, fr, de, cs} subset of the Medical domain. Some models are also included in the main paper.

ID	Model	All	In→in	Out→in	In→out	Out→out
	Oracles					
(10)	Finetune (all langs)	95%	95%	95%	95%	95%
(3)	FT (all langs & domains) + domain tags	95%	95%	96%	95%	95%
	Baselines					
(1)	Base (En-centric)	91%	93%	92%	91%	89%
(11)	**(1)** + ParaCrawl LA	95%	96%	96%	95%	95%
	Straightforward Methods					
(12)	**(1)** + Domain adapters only	40%	95%	93%	7%	11%
(13)	Freeze LA + enc. & dec. DA	81%	95%	93%	71%	76%
(16)	FT (all domains) + dom. tags	25%	96%	55%	1%	2%
	Improving Off-target Translation					
(17)	**(16)** + ParaCrawl	93%	95%	93%	93%	92%
(34)	Unfreeze LA	94%	95%	95%	93%	95%
(35)	**(34)** + BT	94%	96%	95%	95%	94%
(15)	Freeze LA + dec. DA	84%	95%	95%	77%	77%
(20)	**(15)** + BT	94%	95%	95%	94%	94%
(36)	**(15)** + BT + DADrop	94%	95%	95%	94%	94%
(14)	Freeze LA + enc. DA	90%	95%	93%	89%	88%
(19)	**(14)** + BT	90%	96%	94%	94%	83%
(37)	**(14)** + BT + DADrop	93%	95%	95%	94%	91%
(38)	Freeze LA + enc. first 3 layers DA	59%	95%	93%	35%	43%
(39)	Freeze LA + enc. last 3 layers DA	88%	95%	94%	83%	86%
(21)	**(13)** + DADrop	86%	95%	93%	82%	82%
(18)	**(13)** + BT	91%	95%	95%	94%	85%
(22)	**(13)** + BT + DADrop	93%	95%	95%	94%	91%
(40)	**(13)** + BT + MAD-X style	89%	95%	95%	92%	80%
(23)	Unfreeze LA + dec. DA	36%	96%	95%	1%	2%
(24)	**(23)** + DADrop	90%	95%	95%	82%	92%
(25)	**(23)** + DADrop + BT	94%	95%	95%	94%	94%

Table 8: On-target translation percentages of various models trained on the {en, fr, de, cs} subset of the Medical domain.

ID	Model	All	In→in	Out→in	In→out	Out→out
	Oracles					
(10)	Finetune (all langs)	.461	.437	.427	.487	.477
(3)	FT (all langs & domains) + domain tags	.433	.423	.409	.455	.438
	Baselines					
(1)	Base (En-centric)	.300	.307	.299	.306	.294
(11)	**(1)** + ParaCrawl LA	.334	.330	.328	.340	.335
	Straightforward Methods					
(12)	**(1)** + Domain adapters only	.246	.451	.349	.163	.150
(13)	Freeze LA + enc. & dec. DA	.165	.449	.137	.144	.089
	Improving Off-target Translation					
(16)	FT (all dom.) + dom. tags	.166	.436	.162	.143	.081
(17)	FT (all dom. + ParaCrawl) + dom. tags	.359	.410	.375	.351	.332
(34)	Unfreeze LA	.352	.454	.351	.322	.335
(15)	Freeze LA + dec. DA	.304	.404	**.385**	.249	.244
(41)	**(15)** + Mono data	.355	.390	.373	.342	.336
(20)	**(15)** + BT	.381	.399	.371	.387	.375
(36)	**(15)** + BT + DADrop	**.382**	.402	.373	.388	**.376**
(14)	Freeze LA + enc. DA	.319	.438	.328	.315	.266
(42)	**(14)** + Mono data	.347	.410	.338	.353	.324
(19)	**(14)** + BT	.365	.432	.353	.385	.330
(37)	**(14)** + BT + DADrop	.368	.425	.354	**.394**	.336
(18)	**(13)** + BT	.374	.434	.366	**.394**	.341
(22)	**(13)** + BT + DADrop	.381	.436	.369	.406	.349
(23)	Unfreeze LA + dec. DA	.224	.457	.351	.088	.138
(24)	**(23)** + DADrop	.339	**.458**	.354	.288	.320
(43)	Multi-domain dec. DA	.326	.403	.360	.304	.285
(44)	Multi-domain enc. DA	.337	.412	.360	.327	.297
(45)	Multi-domain enc. & dec. DA	.327	.417	.369	.302	.279

Table 9: chrF score of various models trained on the {en, fr, de, cs} subset of the Koran domain. LA = language adapters, DA = domain adapters. 'Out→in' is the average score when translating from an out-of-domain source language into {en, fr, de, cs}. 'In→out' corresponds to when the out-of-domain language is the target language. 'In→in' refers to average score when source and target are in the set {en, fr, de, cs}. 'Out→Out' is the average score when both the source and target language are unseen during domain adaptation. 'Mono data' refers to adding copied monolingual data for out-of-domain languages, and additionally multiparallel ParaCrawl data in small amounts.

ID	Model	All	In→in	Out→in	In→out	Out→out
(13)	Freeze LA + enc. & dec. DA	.309	.491	.362	.267	.229
(21)	**(13)** + DADrop	.311	.490	.360	.270	.233
(34)	Unfreeze LA	.357	.515	.395	.303	.307
(35)	**(34)** + BT	.372	.529	.364	.370	.319
(15)	Freeze LA + dec. DA	.332	.463	**.418**	.268	.262
(20)	**(15)** + BT	**.382**	.460	.408	.362	**.345**
(14)	Freeze LA + enc. DA	.320	.491	.345	.296	.249
(19)	**(14)** + BT	.359	.492	.374	.354	.298
(23)	Unfreeze LA + dec. DA	.322	.519	.399	.216	.267
(24)	**(23)** + DADrop	.353	.515	.399	.291	.303
(25)	**(23)** + DADrop + BT	.377	**.522**	.372	**.373**	.327
(18)	**(13)** + BT	.368	.490	.388	.363	.308
(22)	**(18)** + DADrop	.373	.494	.389	.369	.314

Table 10: chrF score of various models trained on the mostly romance language {en, fr, it, es} subset of the Koran domain. LA = language adapters, DA = domain adapters. 'Out→in' is the average score when translating from an out-of-domain source language into {en, fr, it, es}. 'In→out' corresponds to when the out-of-domain language is the target language. 'In→in' refers to average score when source and target are in the set {en, fr, it, es}. 'Out→Out' is the average score when both the source and target language are unseen during domain adaptation.

ID	Model	IT	Koran	Medical	Subtitles	Law
(46)	No fine-tuning	35.3	14.8	38.1	26.8	42.4
(47)	Fine-tuned	43.8	22.7	53	30.9	57.9
(48)	Enc. + dec. adapters ($d = 1024$)	42.9	21.8	51.7	30.5	56
(49)	**(48)** + MAD-X style	40.6	19.3	48.8	29.8	54.3
(50)	Dec. adapters ($d = 2048$)	42.1	19.8	50.5	29.7	55.1
(51)	Enc. adapters ($d = 2048$)	42.4	21.5	51.9	30.1	56.1
(52)	Last 3 encoder layers only ($d = 4096$)	42.9	21.1	52.1	30.1	56
(53)	First 3 encoder layers only ($d = 4096$)	42.2	20	50.1	28.5	54.9

Table 11: BLEU scores of various domain adaptation strategies for a German → English bilingual model. ($d = N$) refers to adapters with a bottleneck dimension of size N.

ID	Model	IT	Medical	Koran	Subtitles	Law
(54)	No fine-tuning	35.3	14.8	38.1	26.8	42.4
(55)	Finetuned	43.8	22.7	53	30.9	57.9
(56)	Enc. + dec. adapters (d=64)	40	18.7	47.3	29.4	51.5
(57)	Dec. adapters (d=128)	39	17.5	46	28.8	50.6
(58)	Enc. adapters (d=128)	40	18.9	47.3	29.2	51.5
(59)	Last 3 encoder layers only (d=256)	40	19	47.3	29	51.1
(60)	First 3 encoder layers only (d=256)	39.5	18	46	28.8	49.5

Table 12: BLEU scores of various domain adaptation strategies for a German → English bilingual model. ($d = N$) refers to adapters with a bottleneck dimension of size N.

5.6 Contribution and impact

This paper has 14 citations according to Google Scholar as of 2023-04-04.

Our paper builds on the paradigm of keeping language and 'task' information separate, originally introduced by Pfeiffer et al. (2020) for multilingual classification tasks. Concurrent work by Üstün et al. (2020) trains language adapters for mBART using a denoising objective, and then fine-tunes the cross-attention parameters for machine translation on a subset of language directions, enabling better zero-shot generalization to new language pairs. Lai et al. (2022) employ a similar technique, training language adapters (again for mBART) with a denoising objective, then fine-tuning cross-attention parameters on English data in a particular *style* of text (such as formal or informal) with English adapters activated. They can then swap out the English adapters for other languages to achieve zero-shot cross-lingual style transfer.

While our paper was focused on composing adapter modules, we found that domain tags represented a strong baseline. Vu et al. (2022) also tackled the problem of cross-lingual transfer and multi-lingual, multi-domain machine translation, and explore additional tricks to make domain tags work better. They start from the approach of composing language and domain adapters, and confirm they work well when dealing with unseen source languages and known target languages. However they find adding domain tags and target language to the *encoder*, and adding an auxiliary classifier that classifiers encoder representations as belonging to a particular domain, improves performance on *unseen* target languages.

Chapter 6

When does Parameter-Efficient Transfer Learning Work for Machine Translation?

Chapter 5 tackled the problem of adapting pre-trained models to machine translation in a parameter-efficient manner. We discovered some limitations in this setting compared to classification. In section 5.1 we saw that tuning small adapter models was not enough to achieve good performance on machine translation when starting from a pre-trained model, mBART, that was trained on a multilingual denoising objective rather than machine translation data. In section 5.4 we started from a model pre-trained on machine translation, but tackled the more difficult task of multilingual, multi-domain MT, with a focus on cross-lingual transfer. For convenience, that analysis focused on a relatively small pre-trained model, that was only trained on European languages.

The paper presented in this chapter aims to more comprehensively answer some of the questions that arose from the previous chapter. How do changes in fine-tuning language pair, pre-training objective, or PEFT architecture effect performance? We are better able to explore these questions due to the availability of new pre-trained models (specifically M2M (Fan et al., 2020), a large transformer encoder-decoder model pre-trained on machine translation data scraped from the web, with 100 languages represented). We perform control experiments where we vary language pair, fine-tuning dataset size or pre-training objective while keeping other factors constant.

One goal was to provide an up-to-date analysis of PEFT methods for MT, using current PEFT architectures and pre-trained models, similar to Chapter 4 (which focused on English classification tasks). We find challenges for PEFT methods, such as them struggling when used with the mBART pre-trained model, and for distantly related language pairs. But we also see that these methods can outperform full fine-tuning for small dataset sizes, and that they work well as pre-trained model size scales up. Thus we remain positive about PEFT methods being relevant as the field evolves, and we comment on potential future directions in section 6.2.

6.1 The paper

This section presents the paper *When does Parameter-Efficient Transfer Learning Work for Machine Translation?* (Üstün and Cooper Stickland, 2022).

The paper was initially published as a preprint on arXiv in May 2022. Then, it was accepted for

publication at the conference *Empirical Methods in Natural Language Processing (EMNLP)* in December 2022.

Our key findings were:

- The simple adapter architecture outperforms the alternatives we tested (prefix-tuning, BitFit and tuning cross-attention parameters), for most parameter 'budgets', except for the very smallest budgets where prefix-tuning works better.
- Language directions with distantly related languages (e.g. English and Korean) require more parameters to be tuned to match full fine-tuning performance. In other words PEFT performance is negatively correlated with fine-tuning language pair distance.
- As expected, for a given parameter budget using M2M (where the pre-training objective is MT) gives you much better performance than mBART.
- As we decrease fine-tuning dataset size, PEFT methods catch up to full fine-tuning, and then overtake it.

A note on *Sinkhorn temperature sampling*, an algorithm used to sample language pairs during pre-training for M2M. We mention this algorithm in the paper but do not explain it. Since some languages are overrepresented in typical MT training data (for example English or French), we can balance the training data in some way. If a language L is naively sampled proportional to its prevalence in training, p_L, *temperature sampling* adjusts this to $p_L^{1/T}$, for some 'temperature' T. Thus, lower-resource languages are seen more often. This was the approach used by previous large-scale multilingual translation systems (Arivazhagan et al., 2019). It also is used as a baseline for balancing low-resource tasks with high-resource ones in Chapter 3. In the English-centric translation case, changing the probability of sampling a language changes the probability of the other languages only through the normalization of probabilities.

However, in the many-to-many case, language distributions are more interdependent. For example, some languages are only paired with a subset of other languages or to an overrepresented language. Thus, sampling them will affect the probability of these other languages they are paired with. This strategy thus has no guarantee to produce a balanced distribution between languages. Fan et al. (2020) propose directly sampling a pair of languages from a matrix of pair probabilities such that the marginal distributions of languages corresponds to some target distribution. This means that each row and column of the matrix should sum to the temperature-scaled probability of the corresponding language. This is Sinkhorn temperature sampling.

Author contributions

Asa Cooper Stickland and Ahmet Üstün contributed roughly equally to the paper, with both writing code, running experiments and writing the paper. Asa ran most of the experiments involving M2M and wrote adapter code, whereas Ahmet ran mBART experiments and wrote prefix-tuning code.

When does Parameter-Efficient Transfer Learning Work for Machine Translation?

Ahmet Üstün*
University of Groningen

Asa Cooper Stickland*
University of Edinburgh

Abstract

Parameter-efficient fine-tuning methods (PEFTs) offer the promise of adapting large pre-trained models while only tuning a small number of parameters. They have been shown to be competitive with full model fine-tuning for many downstream tasks. However, prior work indicates that PEFTs may not work as well for machine translation (MT), and there is no comprehensive study showing when PEFTs work for MT. We conduct a comprehensive empirical study of PEFTs for MT, considering (1) various parameter budgets, (2) a diverse set of language-pairs, and (3) different pre-trained models. We find that 'adapters', in which small feed-forward networks are added after every layer, are indeed on par with full model fine-tuning when the parameter budget corresponds to 10% of total model parameters. Nevertheless, as the number of tuned parameters decreases, the performance of PEFTs decreases. The magnitude of this decrease depends on the language pair, with PEFTs particularly struggling for distantly related language-pairs. We find that using PEFTs with a larger pre-trained model outperforms full fine-tuning with a smaller model, and for smaller training data sizes, PEFTs outperform full fine-tuning for the same pre-trained model.[1]

1 Introduction

There has been enormous progress on scaling up neural machine translation (NMT) in the recent years, resulting in 'massively multilingual' models that are capable of translating across many languages (Bapna et al., 2022). Most successful applications rely on sequence-to-sequence pre-training that (1) leverages web-scale monolingual data with a masking objective to build a multilingual backbone (parent) model (Liu et al., 2020; Song et al., 2019), or (2) directly targets a many-to-many NMT system by mining parallel corpora (Fan et al., 2020).

Standard practice is to fine-tune every parameter of a particular a pre-trained model to specialize it to a language pair (or domain) of interest (Zoph et al., 2016; Neubig and Hu, 2018). However, if we require specialization to many language pairs or domains, the storage and time costs of full fine-tuning may become prohibitive. Moreover, as models grow ever larger, more efficient methods become attractive.

As an alternative to full model fine-tuning, several parameter-efficient fine-tuning methods (**PEFTs**) have been proposed. Such methods only fine-tune a small number of parameters, reducing storage cost, and avoid calculating the gradients for every model parameter, reducing training time and memory cost. Examples include adapters (Houlsby et al., 2019; Bapna and Firat, 2019) and prefix-tuning (Li and Liang, 2021), which introduce a few extra parameters to fine-tune, keeping the pre-trained model fixed. Others like BitFit (Zaken et al., 2021) tune only the bias vectors of the backbone model and similarly Gheini et al. (2021) update only cross-attention layers.

PEFTs can produce results that are competitive with full fine-tuning. For instance, adapters can match full fine-tuning performance on the GLUE benchmark using only 2-4% additional parameters (Houlsby et al., 2019). However their potential for MT has not been fully explored. Prior studies indicate that PEFTs designed for classification tasks can fail for MT (Stickland et al., 2021a), and it is not known how source and target language characteristics affect PEFTs' performance.

In this work, we provide a comprehensive analysis of PEFTs for MT. For our analysis, we consider: (1) different pre-trained models which vary in size

* Both authors contributed the paper equally and the order is determined by coin flip.

[1] Our code and scripts for reproducing the experiments are available at https://github.com/ahmetustun/fairseq

from 484 million to 1.2 billion total parameters, (2) several PEFTs, and (3) typographically and geographically diverse languages. Moreover, we vary the number of tuned parameters, resulting in different parameter 'budgets', ranging from 0.03% to 10% of total model parameters. Our main research questions are:

RQ1: For a given parameter budget, which PEFT works best?

RQ2: How does language similarity affect the performance of PEFTs for different parameter 'budgets'?

RQ3: How does (i) the pre-training objective, and (ii) the size of the parent model affect the performance of PEFTs?

RQ4: Do PEFTs work better than fine-tuning for small dataset sizes?

Key Findings **1)** We found methods which introduce new parameters to a pre-trained model, namely adapters and prefix tuning, give us the best performance (§ 5.1). As we increase the number of new parameters, adapters retain good performance, while prefix-tuning falls behind. **2)** We found a large variation in PEFTs' performance across language pairs. Specifically, the distance between the source and target languages is negatively correlated with performance, especially for methods tuning the smallest number of parameters and methods tuning a subset of existing parameters (like bias terms or cross attention) (§ 5.2). **3)** We observe that increasing model size, but keeping the same number of fine-tuned parameters, substantially increases MT performance (§ 5.3). Finally, **4)** we observe that adapters perform better than full fine-tuning for small datasets, with the advantage for adapters increasing as dataset size gets smaller (§ 5.4).

2 Background

This section briefly describes the two multilingual pre-trained models that we focus on in this work, namely mBART and M2M-100.

Multilingual Denoising Pre-training Multilingual BART, mBART (Liu et al., 2020), is a sequence-to-sequence transformer model (Vaswani et al., 2017) that consists of an encoder and an autoregressive decoder. It is pre-trained with a *denoising* objective, reconstructing a document from a noisy version. mBART uses span masking and sentence permutation to noise the original document. It consists of 12 encoder and 12 decoder layers, with hidden dimension of 1024 and 16 attention heads. mBART is trained entirely on monolingual data that includes multiple languages and it has a large multilingual vocabulary of 250k tokens. In our experiments, we use mBART-50 (Tang et al., 2020) which was pre-trained on 50 languages.

Many-to-Many Multilingual MT The M2M-100 model (Fan et al., 2020) is a many-to-many multilingual translation system that is pre-trained on a large-scale parallel dataset for 100 languages and 100×99 translation directions. This dataset is automatically constructed with a novel data mining method based on language similarities and back-translation. The model is trained in a many-to-many fashion, balancing languages using *sinkhorn* temperature sampling. In our experiments, we use the base size M2M-100 with 484M parameters that consists of 12 encoder and 12 decoder layers, hidden dimension of 1024 and feedforward dimension of 4096. To study the effect of model size, we also use the medium size M2M-100 with 1.2B parameters, which has 24 encoder and 24 decoder layers, and feedforward dimension of 8192. Both models have a multilingual vocabulary of 128K unique tokens that are distributed across 100 languages with temperature sampling.

3 Parameter Efficient Fine-tuning Methods

All of our experiments fall under the umbrella of specialising a pre-trained sequence-to-sequence transformer model for MT of a particular language pair, with source language x and target language y. If the pre-training task was MT, and x and y were included, then a lower bound will be simply applying the pre-trained model without any changes. Conversely an upper bound is fine-tuning 100% of the pre-trained model parameters ('full fine-tuning'). In between full fine-tuning and directly using the pre-trained model, we consider the following parameter-efficient fine-tuning methods (PEFTs) in this work:

Adapter-tuning (Houlsby et al., 2019) 'Adapter layers' are lightweight, learnable units inserted between transformer layers. They typically take the form of a feedforward network inserted as the final operation in a transformer layer. Formally, we follow the architecture introduced by Bapna and Firat

(2019) for MT:

$$\mathrm{A}_\ell(\mathbf{h}^\ell) = W_\mathrm{u}^T \cdot f(W_\mathrm{d}^T \mathrm{LN}(\mathbf{h}^\ell) + \mathbf{b}_\mathrm{d}^\ell) + \mathbf{b}_\mathrm{u}^\ell, \quad (1)$$

where an adapter module A_ℓ at layer ℓ consists of a layer-normalization LN of the input $h^\ell \in \mathcal{R}^d$, followed by a down-projection $W_\mathrm{d} \in \mathcal{R}^{d \times b}$ with bottleneck dimension b, a non-linear function $f(\cdot)$ and an up projection $W_\mathrm{u} \in \mathcal{R}^{b \times d}$. Finally, a residual connection with input h^ℓ is added to the output of the adapter: $\mathbf{h}^\ell \rightarrow \mathrm{A}_\ell(\mathbf{h}^\ell) + \mathbf{h}^\ell$. We write 'adapter-$b$' to mean adapters with bottleneck dimension b throughout this work.

Prefix-tuning (Li and Liang, 2021) prepends a sequence of continuous task-specific vectors ('prefixes') to the model input, in analogy to natural language prompts (e.g. 'translate this sentence:') which the transformer can attend to, but the prefix consists entirely of free parameters. For each transformer layer, the prefix is replaced with a new set of vectors, increasing expressiveness. Concretely, we replace token embeddings by

$$E_p = \mathrm{Concat}(V^0, E), \quad (2)$$

with $E \in \mathcal{R}^{L \times d}$ the original token embeddings packed into a matrix, $V^0 \in \mathcal{R}^{p \times d}$ the prefix vectors, and L the original sequence length, p the prefix length and d model dimension. Before transformer layer ℓ we additionally set the first p hidden states to a new prefix vector, i.e. $H^\ell[:p,:] = V^\ell$ with $H \in \mathcal{R}^{(L+p) \times d}$ the hidden states and $V^\ell \in \mathcal{R}^{p \times d}$.

BitFit (Zaken et al., 2021) Bias term fine-tuning was introduced in the context of fine-tuning BERT for classification tasks, and consists of training only the bias terms and the task-specific classification layer. For MT we additionally fine-tune all decoder bias terms, and do not need the classification head. We introduce a simple improvement to BitFit, based on replacing redundant parameters with ones that increase expressiveness. Note that BitFit fine-tunes bias parameters in layer-norm (LN) modules (Ba et al., 2016), since the layer-norm contains the following affine transformation:

$$\mathrm{LN}_\mathrm{aff}^\ell(\mathbf{z}^\ell) = \gamma \odot \mathbf{z}^\ell + \beta \quad (3)$$

where $\mathbf{z}^\ell$ is the normalized input after a residual connection. $\gamma, \beta \in \mathcal{R}^d$ are learnable weights and the bias parameters of the LN module. For the standard transformer model, the LN module is always followed by a matrix multiplication plus a bias term i.e. $W_m^\ell \cdot \mathrm{LN}_\mathrm{aff}^\ell(\mathbf{z}^\ell) + b_m^\ell = W_m^\ell \cdot \gamma \odot \mathbf{z}^\ell + W_m^\ell \cdot \beta + b_m^\ell$. Notice the same space of functions is available by *only* updating the b_m^ℓ term in $W_m^\ell \cdot \beta + b_m^\ell$. We simply switch to updating γ instead of β, i.e. unfreezing the LN weight and freezing the bias, in order to increase expressiveness (confirmed empirically in § 5.1). We use this version of BitFit throughout this work unless stated otherwise.

	mBART		M2M-100	
	it→en	tr→en	it→en	tr→en
Full FT	38.2	31.7	36.6	30.1
X-attention	34.8	27.0	36.1	29.2
Adapter (b=1024)	<u>38.0</u>	<u>30.6</u>	36.3	<u>30.0</u>
Prefix (p=13)	29.7	20.3	32.7	26.7
BitFit (LN-bias)	29.3	19.9	32.4	26.2
BitFit (LN-weights)	<u>30.5</u>	21.1	32.6	26.4
Adapter (b=5)	29.9	<u>21.9</u>	<u>33.2</u>	<u>26.9</u>
Prefix (p=5)	<u>28.4</u>	<u>19.1</u>	32.4	26.3
Adapter (b=1)	27.8	15.3	32.5	26.5

Table 1: For a given parameter budget, which method works best (RQ1)? BLEU scores for it→en and tr→en when different fine-tuning methods used for mBART and M2M-100. Each block consists of methods that update approximately the same number of parameters. We underline results which are significantly (p<0.05) best within a block w.r.t. *paired bootstrap resampling*. chrF scores for these experiments are shown in Appendix C.

X-attention Tuning (Gheini et al., 2021) refers to fine-tuning only cross-attention (X-attention) and corresponding layer-norm parameters located in each decoder layer of a transformer model. This method is based on the importance of cross-attention for MT.

4 Experiments

Datasets We conduct experiments with a selection of 12 typologically and geographically diverse languages, paired with English. In our experiments, we fine-tune the pre-trained model on only one language pair and translation direction at a time (e.g. Italian → English). The parallel data for all languages is from TED talks in order to factor out the impact of the domain differences (except Finnish and Estonian which we only use for a separate control experiment). To pick these languages, we consider variation in language families and scripts. More details of the datasets are given in Appendix A.

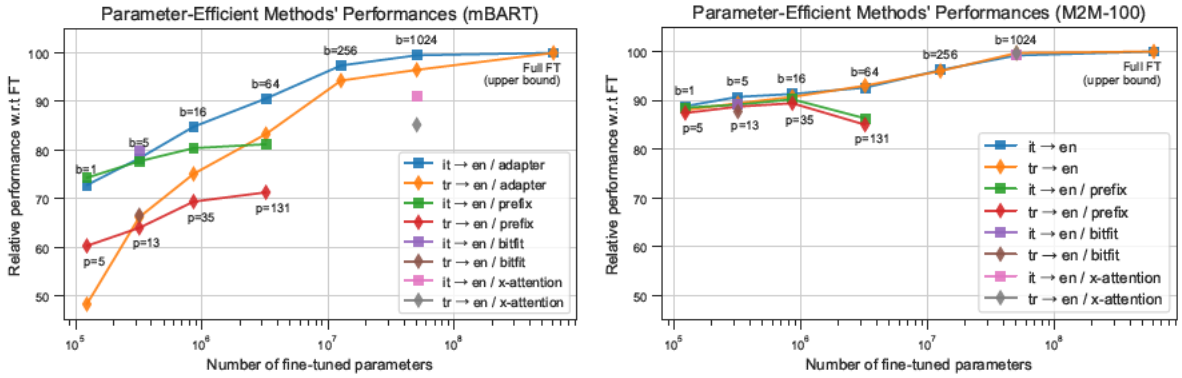


Figure 1: For increasing parameter budget, does prefix-tuning or adapters work best (RQ1)? We show relative MT performance over full fine-tuning vs. number of fine-tuned parameters for mBART and M2M-100. b and p refer to adapter bottleneck dimension and prefix length respectively. Due to the large effective sequence length, we limit prefix-tuning experiments.

Experimental Settings We used mBART-50 (Liu et al., 2020; Tang et al., 2020) and M2M-100 (Fan et al., 2020) as our multilingual pre-trained models, and all the languages we experiment with are included in their pre-training data. mBART needs to learn machine translation with parallel data, but M2M-100 can also be used without fine-tuning, since it is initially pre-trained for MT (see § 2). We conduct experiments with both the base and the medium size M2M-100, to measure the impact of parent model size.

For all fine-tuning methods, we fine-tuned models with a maximum learning rate of 1e-4 with 2500 warm-up steps for 100K training updates. We picked the best model based on dev set perplexity. We used a maximum batch size of 1024 tokens for mBART and 600 tokens for M2M-100, with a gradient accumulation step (*update-frequency*) of 2 for both models. All experiments are performed with the fairseq (Ott et al., 2019) library. Additional details including dataset splits are in Appendix A.

We use BLEU scores to estimate MT quality, calculated from Sacrebleu[2] (Post, 2018). To compare fine-tuning methods across different languages, we often report **relative performance** with respect to full fine-tuning (FT) for each language by calculating the ratio of each method's BLEU score w.r.t. the full FT BLEU score.[3] On the recommendation of Marie et al. (2021) we report chrF (Popović, 2015) in Appendix C for each fine-tuning method.

Parameter Budget Selection In order to fairly compare different methods, we selected a series of parameter 'budgets', and adjusted the settings of each method such that they update the same number of parameters. To determine the parameter budgets, we used the number of trainable parameters for the cross-attention update and BitFit since these numbers are constant (Unlike adapters and prefix-tuning, where we have an adjustable bottleneck dimension). Additionally, when comparing adapters and prefix-tuning, we start from the parameter size of the smallest adapter where the bottleneck dimension is 1.[4]

5 Results and Discussion

In this section, we first compare the performance of various PEFTs on two language directions for different parameter budgets § 5.1. We then select a subset of these methods to test on ten language directions, in order to evaluate the effect of language similarity on the performance of PEFTs § 5.2. We use these results to explore the effect of parent model pre-training § 5.3 and parent model size § 5.3. We noticed that on the language directions with the smallest dataset size, adapter methods outperformed full fine-tuning, and therefore conducted control experiments showing that as dataset size decreases, adapters outperform full fine-tuning by a larger margin.

[2]Sacrebleu signature (BLEU): nrefs:1|case:mixed|eff:no|tok:13a|smooth:exp|version:2.0.0
For the significance test we used `-paired-bs` flag.

[3]BLEU scores for each direction are given in Appendix C

[4]Each block corresponds to parameter budgets of approximately 50m, 320k, and 120k trainable parameters, representing {X-attention, Adapter-1024}, {BitFit, Adapter-5, Prefix-13}, and {Adapter-1, Prefix-5} respectively.

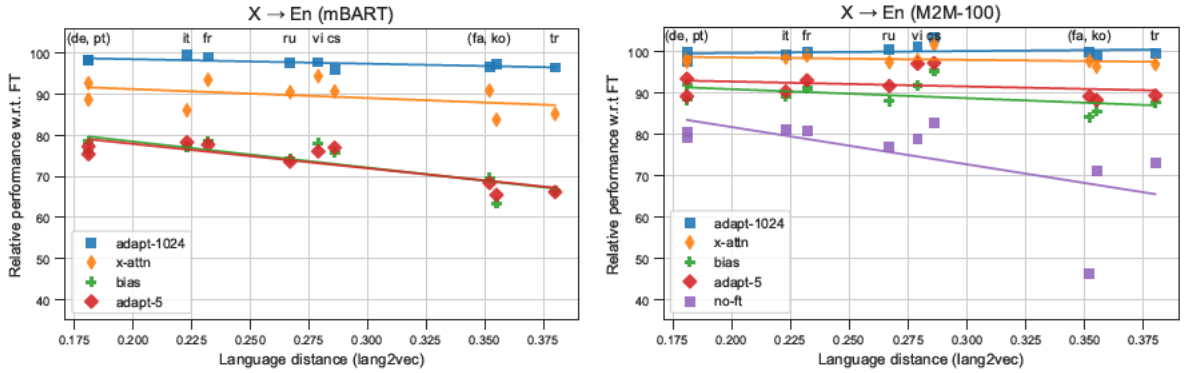


Figure 2: How does language similarity affect relative performance in x→en with respect to full fine-tuning (%) for PEFTs (RQ2)? Trend lines show the correlation between performance of PEFTs and language distance.

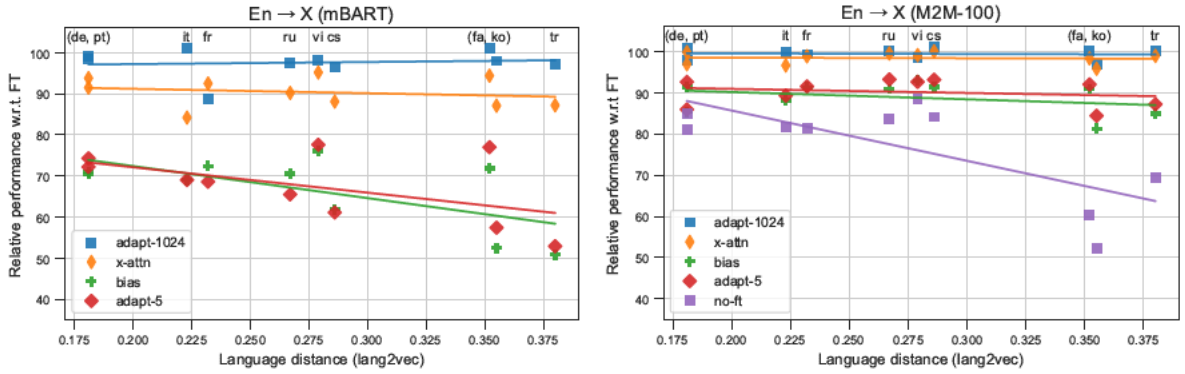


Figure 3: How does language similarity affect relative performance in en→x with respect to full fine-tuning (%) for PEFTs (RQ2)? Trend lines show the correlation between performance of PEFTs and language distance.

5.1 RQ1: Comparing fine-tuning methods

Table 1 shows the **performance** of PEFTs in terms of BLEU score for it→en and tr→en. In the table, each block (separated with a dashed line) consists of PEFTs with approximately the same number of updated parameters. Adapters outperform other methods for almost all parameter budgets for both mBART and M2M-100, except the smallest budget of 120k updated parameters. In this block, prefix-tuning (prefix-5) performs better than adapters for mBART. However, when the fine-tuned parameter count increases, as shown in Figure 1, prefix-tuning quickly falls behind adapters, confirming previous findings (He et al., 2021a). Furthermore, in terms of **training speed/memory cost**, prefix-tuning slows down training relative to adapters, and imposes a significant memory cost due to a large effective sequence length; see also Appendix B.[5]

As for the methods that fine-tune existing parameters, both BitFit and X-attention performs worse than adapters in most cases. Averaging across 10 language pairs, adapters still outperform BitFit for both parent models (Figure 5). However, we confirm that our method of tuning layer norm weights rather than biases improves BitFit, see Table 1.

5.2 RQ2: Impact of language relatedness

In order to evaluate how language similarity between translation pairs affects the performance of different PEFTs, we extend our experiments to 10 languages paired with English (x→en, en→x), representing a diverse set of linguistic typology. Figure 2 and 3 show performance w.r.t. full fine-tuning, for both mBART and M2M.

We found that similarity between source and target languages impacts the performance of PEFTs, with distantly related languages (e.g. English and Korean) leading to lower performance for the methods with a small number of updated parameters such as BitFit and adapter-5. And so when translating between distantly related languages, we need to tune more parameters to match full fine-tuning and get the most out of the parent model.

[5]Prefix-13 causes a 30% slow-down in training speed relative to adapter-5.

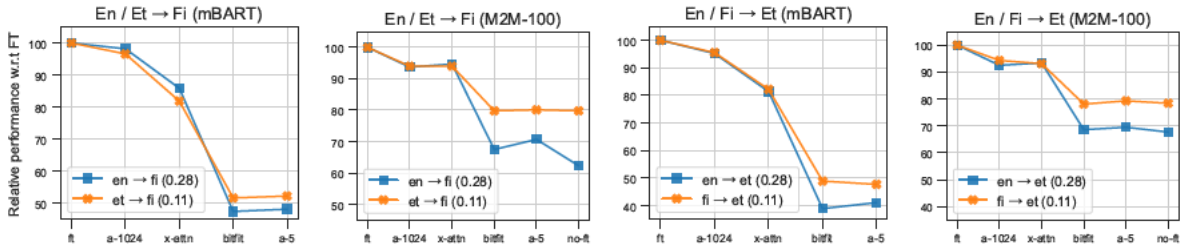


Figure 4: Decrease in relative performance (%) over full fine-tuning as the number of updated parameters decreases for translating into Finnish and Estonian with different source language (en, et, fi). `Lang2vec` distances for en↔fi, en↔et, and fi↔et are 0.28, 0.28 and 0.11 respectively.

More concretely, relative performance w.r.t. full FT is negatively correlated with language distance measured by `lang2vec`[6]. These correlations are stronger for mBART than M2M. Methods which tune existing parameters (X-attention and BitFit) and M2M with no fine-tuning show higher correlation than adapters with similar parameter budgets; see Table 5. One explanation is that adding parameters, and therefore increasing model capacity with adapters is beneficial for overcoming the difficulty of translating distant languages.

We provide correlation results with more fine-grained measures of language distance, namely syntactical, phonological and geographical distances in Appendix D. For the first two distances, we observe a similar trend: as the distance between source and target language increases, BitFit and small adapters do not perform as well (the negative correlation is stronger). Generally the syntactic features produced a larger negative correlation than the phonological features, with the exception of M2M plus PEFTs for en→x. However, in terms of geographic distance, we do not observe a particular trend.

To investigate whether our findings extend beyond English-centric settings, we designed another set of experiments. We picked 3 languages from MultiParaCrawl, Finnish, Estonian and English, where Finnish and Estonian are from the same language family and typologically similar. We measure translation performance into Finish from Estonian and English, for different fine-tuning methods, and similarly for translation into Estonian. Figure 4 shows results for both mBART and M2M-100.

As shown in the first two plots, when translating into Finnish, Estonian as the source language gives an advantage over English for BitFit and adapter-5 (This advantage is higher in M2M-100 than mBART). Likewise, for translation into Estonian, as the number of trainable parameters decreases, relative MT performance drops less when Finnish is the source language compared to English, for both parent models. Thus, when the source and target languages are typologically similar, PEFTs make better use of the parent model.

	mBART		M2M-100	
	x→en	en→x	x→en	en→x
Adapter (b=1024)	*-0.43*	*0.11*	*0.23*	*-0.07*
X-attention	-0.69	*-0.21*	*-0.28*	*-0.07*
Adapter (b=5)	-0.85	*-0.53*	*-0.26*	*-0.22*
BitFit	-0.84	-0.64	*-0.47*	*-0.33*
No FT	-	-	-0.60	-0.72

Table 2: Pearson correlation coefficients between relative performance w.r.t. fine-tuning and language distance. Negative correlation means that relative performance tends to decrease as the distance between source and target language increases. Numbers in italics are not statistically significant (p=0.05).

5.3 RQ3: Impact of parent model

Pre-training Objective Figure 5 shows the overall performances for PETFs aggregated over all languages (x↔en) when the model is initialized with mBART or M2M-100. In general, PEFTs for M2M-100 provides higher *relative* performance than mBART (Fig. 5). This difference is larger when the number of trainable parameters is small (BitFit and adapter-5). While M2M-100 is pre-trained for MT with parallel data, mBART is pre-trained with a (monolingual[7]) denoising objective. Thus, more parameters are required at fine-tuning time to 'learn' the MT task for mBART. Finally, we

[6]lang2vec is a python package based on the URIEL typology database (Littell et al., 2017). For language distance, we compute the cosine distance between typological feature vectors of languages that consists of syntactic, phonological and inventory features (289 features in total).

[7]Although mBART-50 pre-trained on 50 languages, the pre-training objective does not use any cross-lingual signal.

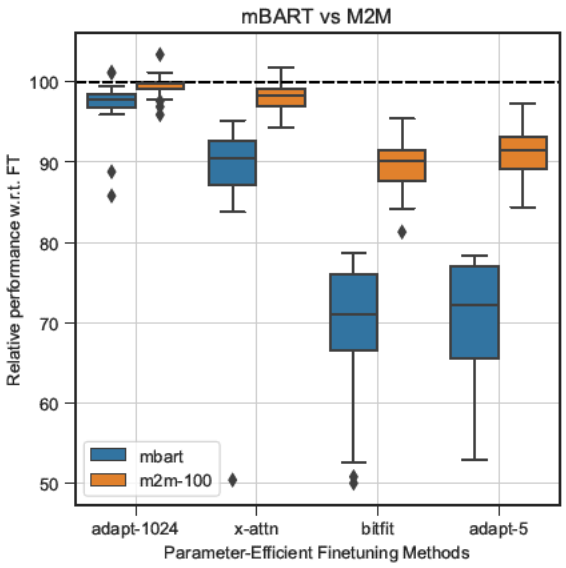


Figure 5: How does the type of parent model (different pre-training objective) affect performance (RQ3)? Statistics of relative performance w.r.t. full fine-tuning (%) for all languages (x↔e) when the model is initialized with mBART or M2M-100. The dashed line refers to full fine-tuning performance.

note mBART results have a higher variance than M2M-100 (see Fig. 5), due to the higher negative correlation with language distance.

Model Size We investigate how parent model size affects the performance of fine-tuning methods, comparing M2M-100's base model (484M) to its medium model (1.2B). Table 3 shows the average performance of full fine-tuning and small-size adapters corresponding to approximately 300K new parameters[8]. No fine-tuning (no FT) results are also shown, representing *lower* bounds.

Predictably, the medium model outperforms the base model across all fine-tuning methods. The magnitude of this improvement is larger when translating into English ($x \rightarrow en$) vs. $x \rightarrow en$, and the increase for small adapters is larger than for other methods. When translating into English, small adapters with the medium model outperform *full fine-tuning* of the base model for most languages despite tuning only 0.03% of its parent model parameters. For $en \rightarrow x$, small adapters are still competitive with full fine-tuning of the base model with almost the same average performance. But for distantly related languages to English (Farsi, Korean and Turkish), adapters' (1.2B) performance falls behind full fine-tuning of the base model.

[8]Both adapter-5 in the base model and adapter-2 in the medium model correspond to roughly the same number of trainable parameters (0.07% of 484M and 0.03% of 1.2B total parameters).

	Model		
	Base (418m)	Med. (1.2b)	Δ BLEU
$en \rightarrow$x			
No FT	21.9	24.3	2.4
Small adapter	24.8	27.4	2.6
Full FT	27.4	28.4	1.0
$x \rightarrow$en			
No FT	26.1	28.5	2.4
Small adapter	31.7	35.3	3.6
Full FT	34.4	36.6	2.2

Table 3: How does parent model size affect performance (RQ3)? Average BLEU score across 10 languages for the base (484m parameters) and medium (1.2 billion parameters) M2M parent models, when tuning all parameters ('full FT'), when tuning small adapters, and when tuning no parameters ('no FT'). We also show the increase in BLEU when moving from the base to the medium model. See Appendix C for individual results.

When it is used without *any* parameter updates ('no FT'), the medium model (while outperforming the base model for no FT) is not competitive with small size adapters for the base model, in either direction ($x \leftrightarrow en$). Furthermore, relative performance w.r.t. full fine-tuning is still negatively correlated with language distance (see Appendix Table 6). Therefore, even at large scales, parameter efficient fine-tuning is useful, taking MT performance to the *upper* bound of a smaller model.

5.4 RQ4: Impact of fine-tuning dataset size

We noticed that for the datasets with the smallest amount of training data (Vietnamese and Czech), PEFTs outperformed full fine-tuning (see Appendix C). We therefore designed a control experiment to test for the effect of the training data size on PEFT' performance, taking a random subset of sizes 2000, 8000, 32000 and 128000 training examples for Italian to English and Turkish to English. We then evaluated full fine-tuning, large adapters (≈50m parameters) and small adapters (≈300k parameters) on each dataset; see Figure 6.

For all models, at the smallest dataset size, large adapters outperformed full fine-tuning, and for M2M full fine-tuning only catches up at 128k examples. For mBART, small adapters lag far behind, indicating they do not provide enough capacity to 'learn' the MT task. For M2M however, small adapters are on a par with larger ones for

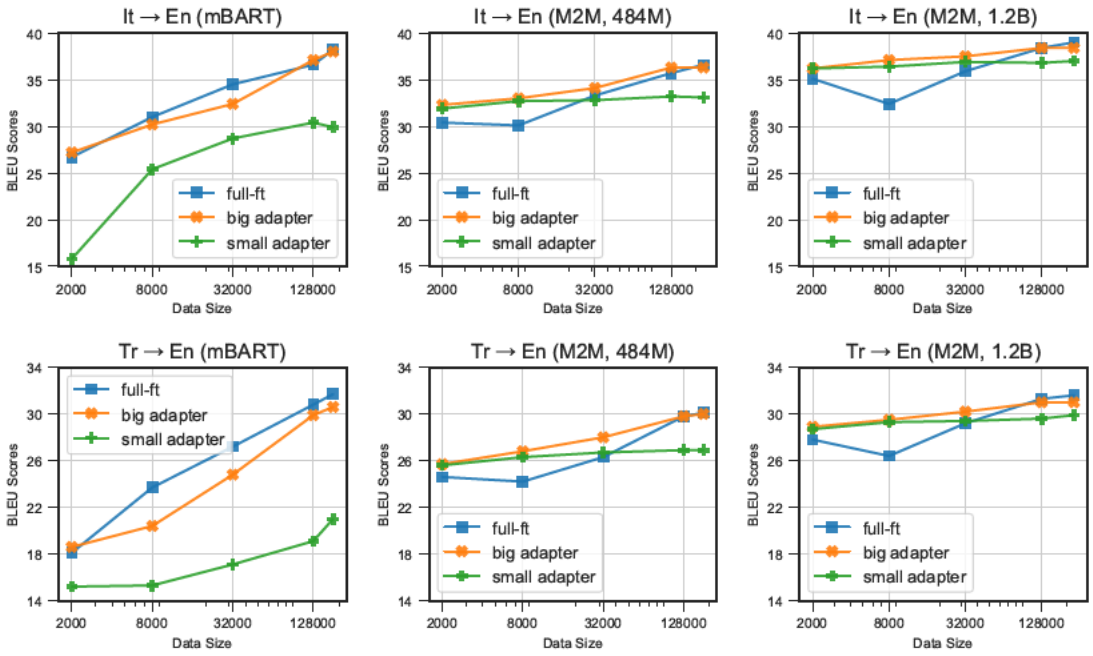


Figure 6: How does the amount of parallel data affect fine-tuning performance (RQ4)? BLEU scores for various subsets of the full training data for Italian to English and Turkish to English, with base (484m parameters) and medium (1.2 billion parameters) M2M and mBART parent models. Big and small adapters have ≈50m and ≈300k parameters respectively for all models.

small dataset sizes, but fall behind as dataset size increases. Again, we believe this is because more capacity is needed to get the most out of larger datasets.

Chen et al. (2022) explore the effect of fine-tuning dataset size for RoBERTa fine-tuned on English NLU tasks, finding PEFTs outperform full fine-tuning for dataset size <1000. Interestingly, for mBART, similarly small dataset sizes are required for outperforming full fine-tuning. However, for M2M, we see adapters outperforming up until dataset sizes of ≈ 128k. Perhaps the 'gap' between RoBERTa's masked language model pre-training objective and the fine-tuning objective is similar to the gap between mBART's pre-training objective and MT, whereas since M2M is pre-trained for MT, leaving the base model unchanged is viable up to larger fine-tuning dataset sizes. We leave further exploration of this to future work. Finally, we observe that full fine-tuning always converges in fewer iterations than the adapter methods, in a result similar to that of Chen et al. (2022).

6 Related Work

PEFTs have been widely used for fine-tuning Transformer models to new tasks, domains or languages. Adapters (Houlsby et al., 2019) have been used in multi-task learning (Stickland and Murray, 2019; Pfeiffer et al., 2021; Karimi Mahabadi et al., 2021), cross-lingual transfer (Üstün et al., 2020; Pfeiffer et al., 2020) and multilingual NMT (Bapna and Firat, 2019; Philip et al., 2020; Stickland et al., 2021b; Üstün et al., 2021). Prefix-tuning (Li and Liang, 2021) and Prompt-tuning (Lester et al., 2021; Qin and Eisner, 2021) (i.e. only using soft prompt tokens without prefix vectors in each layer), have a natural interpretation in terms of virtual tokens. They can be used as task embeddings for inter-task transferability (Vu et al., 2021). LoRA (Hu et al., 2021) injects trainable low-rank matrices into query and value projection matrices of each transformer layer. He et al. (2021a) present a unified framework that integrates the above methods.

Some of these methods have been compared in a controlled setting for English classification tasks (Chen et al., 2022) or only a single language pair (English and Romanian) for MT (He et al., 2021a). Chen et al. (2022) test PEFTs for various English classification tasks and observe that on the tasks with the smallest dataset sizes, PEFTs outperform fine-tuning, but they do not conduct a control experiment varying dataset size and parent model for a single task as we do.

Aspects of efficiency and scale in MT in terms of

inference cost (Berard et al., 2021), vocabulary size (Gowda and May, 2020) data (Gordon et al., 2021), model size (Gordon et al., 2021; Arivazhagan et al., 2019) and number of languages (Arivazhagan et al., 2019) have been explored. Other work aims to improve full FT for domain adaptation by mixing in different data (Chu et al., 2017), regularisation (Miceli Barone et al., 2017) or many other methods (Chu and Wang, 2018; Saunders, 2021). However, none of these works study PEFTs for MT, and we aim to fill this gap.

7 Conclusion

Do PEFTs work for MT? We found that the answer depends on multiple factors: the particular method, the backbone model, the number of tuned parameters and the fine-tuning language pair. Adapters usually have the highest performance out of all PEFTs (§ 5.1), although for the smallest parameter budgets we consider, prefix tuning outperforms adapters for mBART. For large parameter budgets (≈50m parameters) adapters almost recover full fine-tuning performance, and even for lower budgets, if the pre-training task was MT, i.e. M2M-100, adapters can recover >90% of full FT performance. However PEFTs only **outperform** full FT for smaller dataset sizes (§ 5.4), less than around ≈2k examples for mBART and ≈128k for M2M. Future work could explore in detail how the difference between pre-training objective and fine-tuning task affects this phenomenon.

Using PEFT with a larger model (M2M-100 medium size) can outperform full FT of a smaller model (M2M-100 base size). However when translating in the en→x direction where x is distantly related to English e.g. Korean, full FT is superior (§ 5.3). More generally, distantly related language pairs require more parameters to be tuned to get close to full FT, for all methods (§ 5.2).

8 Limitations

Firstly, in this work we do not cover all parameter-efficient fine-tuning methods (or variations on those that we do analyse) such as LoRA (Hu et al., 2021), or mix-and-match adapters (He et al., 2021b). In order to make our analysis compact and clear we center our comparison around simple adapters and prefix-tuning, together with BitFit and updating cross-attention. Secondly, our experiments only cover models with up to around 1 billion parameters due to compute limitations, which does not include the largest models available, such as the 11 billion parameter M2M or mT5 (Xue et al., 2021) models similar to the encoder-decoder models we use in this paper, or much larger autoregressive (and trained largely on English data) language models e.g. Chowdhery et al. (2022). Thirdly, although we attempted to cover a diverse set of languages, we did not explore truly low resource languages, and those not included in the pre-training data of our models (introducing another confounding factor for our language distance analysis), where one would expect even larger performance gaps for PEFTs. However, we do imitate a very-low resource setup by limiting training data size (Section § 5.4). Furthermore, although we attempt to look into PEFTs' performances across languages w.r.t. different distance metrics such as syntax, phonology and geography (Appendix § D), more analysis in terms of fine-grained attributes such as word order or morphology are not provided in our analysis, which we leave for future work. Finally, we use automatic/string-based quality metrics, BLEU and chrF++ (Popović, 2017), rather than pre-trained/neural quality metrics, with the latter often better correlated with human judgements (Kocmi et al., 2021).

Acknowledgements

We would like to thank Iain Murray, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord for valuable comments on a draft of this paper. We also would like to thank the Center for Information Technology of the University of Groningen for providing access to the Peregrine HPC cluster. Asa Cooper Stickland was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

References

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *CoRR*, abs/1907.05019.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization.

Ankur Bapna, Isaac Caswell, Julia Kreutzer, Orhan Firat, Daan van Esch, Aditya Siddhant, Mengmeng Niu, Pallavi Baljekar, Xavier Garcia, Wolfgang Macherey, et al. 2022. Building machine translation systems for the next thousand languages. *arXiv preprint arXiv:2205.03983*.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

Alexandre Berard, Dain Lee, Stephane Clinchant, Kweonwoo Jung, and Vassilina Nikoulina. 2021. Efficient inference for multilingual neural machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8563–8583, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268, Trento, Italy.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022. Revisiting parameter-efficient tuning: Are we really there yet?

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

Chenhui Chu, Raj Dabre, and Sadao Kurohashi. 2017. An empirical comparison of domain adaptation methods for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 385–391, Vancouver, Canada. Association for Computational Linguistics.

Chenhui Chu and Rui Wang. 2018. A survey of domain adaptation for neural machine translation. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1304–1319, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Edouard Grave, Michael Auli, and Armand Joulin. 2020. Beyond english-centric multilingual machine translation. *arXiv preprint*.

Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. Cross-attention is all you need: Adapting pretrained Transformers for machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1765, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mitchell A Gordon, Kevin Duh, and Jared Kaplan. 2021. Data and parameter scaling laws for neural machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5915–5922, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Thamme Gowda and Jonathan May. 2020. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online. Association for Computational Linguistics.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021a. Towards a unified view of parameter-efficient transfer learning. *CoRR*, abs/2110.04366.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021b. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. 2021. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint*

Conference on Natural Language Processing (Volume 1: Long Papers), pages 565–576, Online. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *Proceedings of ICLR*.

Tom Kocmi, Christian Federmann, Roman Grundkiewicz, Marcin Junczys-Dowmunt, Hitokazu Matsushita, and Arul Menezes. 2021. To ship or not to ship: An extensive evaluation of automatic metrics for machine translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 478–494, Online. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Anoop Kunchukuttan, Pratik Mehta, and Pushpak Bhattacharyya. 2018. The IIT Bombay English-Hindi parallel corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

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

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

Patrick Littell, David R. Mortensen, Ke Lin, Katherine Kairis, Carlisle Turner, and Lori Levin. 2017. URIEL and lang2vec: Representing languages as typological, geographical, and phylogenetic vectors. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 8–14.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Benjamin Marie, Atsushi Fujita, and Raphael Rubino. 2021. Scientific credibility of machine translation research: A meta-evaluation of 769 papers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7297–7306, Online. Association for Computational Linguistics.

Antonio Valerio Miceli Barone, Barry Haddow, Ulrich Germann, and Rico Sennrich. 2017. Regularization techniques for fine-tuning in neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1489–1494, Copenhagen, Denmark. Association for Computational Linguistics.

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 875–880.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Jerin Philip, Alexandre Berard, Matthias Gallé, and Laurent Besacier. 2020. Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online. Association for Computational Linguistics.

Maja Popović. 2015. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.

Maja Popović. 2017. chrF++: words helping character n-grams. In *Proceedings of the Second Conference on Machine Translation*, pages 612–618, Copenhagen, Denmark. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Ye Qi, Devendra Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. 2018. When and why are pre-trained word embeddings useful for neural machine translation? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 529–535, New Orleans, Louisiana. Association for Computational Linguistics.

Guanghui Qin and Jason Eisner. 2021. Learning how to ask: Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5203–5212, Online. Association for Computational Linguistics.

Danielle Saunders. 2021. Domain adaptation and multi-domain adaptation for neural machine translation: A survey. *CoRR*, abs/2104.06951.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. In *International Conference on Machine Learning*, pages 5926–5936.

Asa Cooper Stickland, Alexandre Bérard, and Vassilina Nikoulina. 2021a. Multilingual domain adaptation for NMT: decoupling language and domain information with adapters. *Sixth Conference on Machine Translation (WMT2021)*.

Asa Cooper Stickland, Xian Li, and Marjan Ghazvininejad. 2021b. Recipes for adapting pre-trained monolingual and multilingual models to machine translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3440–3453, Online. Association for Computational Linguistics.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995.

Yuqing Tang, Chau Tran, Xian Li, Peng-Jen Chen, Naman Goyal, Vishrav Chaudhary, Jiatao Gu, and Angela Fan. 2020. Multilingual translation with extensible multilingual pretraining and finetuning. *arXiv preprint arXiv:2008.00401*.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Ahmet Üstün, Alexandre Berard, Laurent Besacier, and Matthias Gallé. 2021. Multilingual unsupervised neural machine translation with denoising adapters. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6650–6662, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2021. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. 2016. Transfer learning for low-resource neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1568–1575, Austin, Texas. Association for Computational Linguistics.

A Reproducibility Report

Datasets All datasets that are used in our experimetns are publicly available. We used TED talks (Qi et al., 2018) for (cs, fr, ko, ru, pt, tr, fa)↔en, IWSLT15 and IWSTL17 (Cettolo et al., 2012) for vi↔en and (it, de)↔en respectively, IITB (Kunchukuttan et al., 2018) for hi↔en. Finally, for (en, et, fi) experiments, we randomly sampled 200k

Language	Language family	Dataset source	Train size (k)	Dev size (k)	Test size (k)
Czech **(cs)**	Slavic	TED	103	3.5	3.8
French **(fr)**	Romance	TED	192	4.3	4.9
Korean **(ko)**	Korean	TED	205	4.4	5.6
Russian **(ru)**	Slavic	TED	208	4.8	5.5
Italian **(it)**	Romance	TED	231	0.9	1.6
Portuguese **(pt)**	Romance	TED†	184	4	4.9
Turkish **(tr)**	Turkic	TED	182	4	5
Vietnamese **(vi)**	Austri-Asiatic	TED†	133	1.6	1.3
German **(de)**	Germanic	TED†	206	0.9	1.6
Farsi **(fa)**	Iranian	TED	150	3.9	4.5
Finnish* **(fi)**	Finnic	mParacrawl	200	3	3
Estonian* **(et)**	Finnic	mParacrawl	200	3	3

Table 4: Details of dataset that is used in our experiments. We gather language pairs (x↔en) from TED (Qi et al., 2018) and IWSLT† (Cettolo et al., 2012) that are both compiled from TED talks. '*' indicates a set of separate controlled experiments where we randomly sampled 200k parallel sentences from MultiParacrawl (mParaCrawl) for corresponding language pairs.

parallel sentences for each language-pair from MultiParacrawl by using OPUS (Tiedemann, 2012). Sizes of train, dev and test splits are given in Table 4. All datasets have licenses allowing non-commercial use.

Pre-trained models and Hyper-parameters We used mBART (Liu et al., 2020) that is extended to 50 languages (Tang et al., 2020). For M2M-100 (Fan et al., 2020), we used base- and medium-size models that consist of 484M and 1.2B parameters.

For all experiments we used the hyper-parameters that are reported by Liu et al. (2020) except learning rate. For the learning rate, we follow Üstün et al. (2021) and used maximum of 1e-4 with polynomial learning rate decay, based on their adapter-tuning experiments. We fine-tune models by using 0.3 dropout, 0.2 label smoothing, 2500 warm-up steps for 100K training updates with an early-stopping patience of 10 epochs. We used a maximum batch size of 1024 tokens for mBART and 600 tokens for M2M-100, with a gradient accumulation step (*update-frequency*) of 2 for both models. For full fine-tuning (and not other methods) with the 1.2 billion size M2M model we use the Adafactor optimizer (Shazeer and Stern, 2018) in order to save memory (and use learning rate 5e-5), and otherwise use the Adam optimizer (Kingma and Ba, 2014). We report the result of a single random seed/training run throughout this work whenever we list BLEU scores. All parameter-efficient fine-tuning methods are implemented on top of the Fairseq framework (Ott et al., 2019). We will share our code and scripts to reproduce all experiments.

Computing Budget and Infrastructure All the experiments are conducted using Tesla V100 GPUs with mixed precision (`fp16`). Parameters that are fine-tuned for each model are reported in the experiments section (§ 4). Each individual experiment took 3-10 hours on one GPU depending on the fine-tuning method and the language-pair.

B Prefix-tuning Details

There is relationship between memory cost and training time for prefix-tuning: including virtual tokens in a sentence will increase the effective length of that sentence, and we can either impose additional memory cost for the virtual tokens, or we can reduce the total number of 'real' i.e. natural language as opposed to virtual tokens in each batch. With the latter method we avoid a large memory cost, however the time taken to iterate through a given number of training examples will be longer, since the number of real tokens per batch will be decreased, increasing training time. We use the latter (decreased 'real' tokens) method.

Finally we note that inference speed will decrease as we increase the number of virtual tokens, since the decoder attention needs to attend to virtual tokens, i.e. when decoding token n it will attend to $n-1+p$ previous tokens for prefix length p.

C Additional Results and Metrics

Table 7 shows chrF (Popović, 2017) scores[9] for the experiments comparing different PEFTs on it→en and tr→en (Table 1). These results confirms that

[9]Sacrebleu signature (chrF2++): nrefs:1|case:mixed|eff:yes|nc:6|nw:2|space:no|version:2.0.0

	mBART		M2M-100	
	x→en	en→x	x→en	en→x
syntax				
Adapter (b=1024)	-0.65	*0.17*	*0.14*	*-0.11*
X-attention	*-0.36*	*-0.0*	*-0.34*	*-0.06*
Adapter (b=5)	-0.85	*-0.33*	*-0.26*	*-0.21*
BitFit	-0.80	-0.53	*-0.49*	*-0.28*
No FT	-	-	-0.69	-0.74
phonology				
Adapter (b=1024)	*-0.01*	*-0.24*	*0.01*	-0.71
X-attention	*0.14*	*0.16*	*-0.35*	-0.47
Adapter (b=5)	-0.45	*-0.06*	*-0.28*	-0.42
BitFit	*-0.42*	*-0.13*	*-0.42*	*-0.34*
No FT	-	-	*-0.36*	-0.54
geography				
Adapter (b=1024)	*-0.18*	*0.21*	*0.07*	-0.58
X-attention	*-0.02*	*0.23*	*0.21*	*-0.24*
Adapter (b=5)	*-0.42*	*0.09*	*0.05*	*-0.13*
BitFit	*0.09*	*-0.06*	*-0.29*	*-0.13*
No FT	-	-	*-0.28*	*-0.36*

Table 5: Pearson correlation coefficients between relative performance w.r.t. fine-tuning and syntactic, phonological and geographical language distances. Negative correlation means that relative performance tends to decrease as the distance between source and target language increases. Numbers in italics are not statistically significant (p=0.05).

	en→x	***x→en***
Adapter (b=2)	-0.75	*-0.39*
No fine-tuning	-0.75	*-0.55*

Table 6: Correlation coefficients between language distance and relative performance for the 1.2 billion size M2M model; see also Table 5. Numbers in italics are not statistically significant (p=0.05).

	mBART		M2M-100	
	it→en	tr→en	it→en	tr→en
Full FT	59.4	53.3	58.2	52.6
X-attention	56.6	48.9	57.7	51.6
Adapter (b=1024)	59.2	52.3	57.8	52.2
Prefix (p=13)	52.4	42.8	55.3	49.7
BitFit (LN-bias)	51.8	41.7	55.0	49.3
BitFit (LN-weights)	52.7	42.8	55.1	49.5
Adapter (b=5)	52.4	44.3	55.5	49.9
Prefix (p=5)	51.4	41.4	54.9	49.5
Adapter (b=1)	50.5	36.5	55.0	49.5

Table 7: chrF scores for it→en and tr→en when different fine-tuning methods used for mBART and M2M-100. Each block represents same ratio of updated parameters. We underline results when a model is the significantly best within a block w.r.t. *paired bootstrap resampling* test.

the trends discussed in Section 4 are the same regardless of metric used for MT quality.

In Tables 8, 9 and 10, we show BLEU scores for other experiments presented in the paper only in terms of performance relative to full FT. Additionally we show adapter-1024 and X-attention scores for M2M-100; in general adapter-1024 outperforms X-attention, and both methods come close to full FT performance or slightly outperform it. Note that for M2M, for the two smallest dataset sizes (cs and vi) we see adapter-1024 (and adapter-2 for the medium size M2M) outperforming full fine-tuning, similarly to § 5.4.

In Table 9 we show results of a smaller (40m parameters) transformer model trained from scratch on each dataset separately, with an architecture consisting of 6 encoder and decoder layers, hidden dimension of 512 and feed-forward hidden dimension 1024. We train a unique sentencepiece (Kudo and Richardson, 2018) vocabulary for each dataset, shared between source and target language, of size approximately 16k. Training hyper-parameters were the same as our other models. For the *x→en* direction almost all of our methods based on pre-trained models outperformed the 'from scratch' baseline, however in the *en→x* direction for mBART the most parameter efficient methods sometimes fall short (see e.g. Turkish or French). For translating into Farsi no pre-trained model outperformed the from scratch model, even with full fine-tuning, suggesting a weakness for particularly low resource resource languages like Farsi. Note per-dataset hyper-parameter search would likely improve performance, especially for 'from scratch' results, but we did not attempt this due to computational constraints.

D Additional Correlation Results for Language Distance

Table 5 shows additional correlation coefficient between PEFTs' performances and different language distances: syntax, phonology and geography. Moreover, Table 6 shows the correlation coefficients between language distance and relative performance for the 1.2 billion size M2M model.

M2M-100	No. Params	fa 150k	it 230k	de 208k	ru 208k	ko 205k	fr 192k	pt 184k	tr 182k	vi 133k	cs 103k
en→x											
Full FT	484m	17.6	32.8	32.0	22.0	9.6	41.3	42.1	17.9	33.9	24.5
Full FT (1.2b)	1.2b	17	33.7	33.6	23.8	9.9	42.8	43.1	18.8	35.2	26.3
Adapter (b=1024)	50m	17.6	32.7	31.3	22.0	9.3	41.0	42.4	17.9	33.4	24.8
X-attention	50m	17.3	31.7	31.0	21.9	9.2	40.8	42.0	17.7	33.6	24.5
BitFit	335k	16.0	28.9	27.3	20.0	7.8	37.7	38.5	15.2	31.5	22.4
Adapter (b=5)	320k	16.2	29.3	27.5	20.5	8.1	37.8	39.0	15.6	31.4	22.8
Adapter (b=2; 1.2B)	344k	14.6	32.5	32.1	23.1	8.9	42.2	43.1	16.7	34.6	26.4
No FT (1.2B)	0	9.7	29.6	29.9	21.1	5.5	37.6	39.6	13.2	32.9	24.0
No FT (484M)	0	10.6	26.8	25.9	18.4	5.0	33.6	35.8	12.4	30.0	20.6
x→en											
Full FT	484m	32.3	36.6	37.2	27.8	22.2	43.2	47.9	30.1	34.3	32.8
Full FT (1.2b)	1.2b	36.1	39	39.3	29.9	23.9	45.1	49.8	31.6	35.7	35.3
Adapter (b=1024)	50m	32.3	36.3	36.3	28.0	22	43.2	47.8	30	34.7	33.9
X-attention	50m	31.6	36.1	36.3	27.1	21.4	42.8	47.1	29.2	33.6	33.4
BitFit	335k	27.2	32.6	32.9	24.5	19.0	39.4	44	26.4	31.5	31.3
Adapter (b=5)	320k	28.8	33.1	33.2	25.5	19.6	40.2	44.8	26.9	33.3	31.9
Adapter (b=2; 1.2B)	344k	31.5	37.3	37.7	28.9	22.2	44.0	48.7	29.9	37.5	35.6
No FT (1.2B)	0	14.9	32.5	32.1	24.1	17.6	37.5	42.0	24.2	29.9	30.1
No FT (484m)	0	14.9	29.7	29.5	21.4	15.8	34.9	38.6	22.0	27.1	27.2

Table 8: x↔en results in terms of BLEU for M2M-100 experiments.

mBART	No. Params	fa 150k	it 230k	de 208k	ru 208k	ko 205k	fr 192k	pt 184k	tr 182k	vi 133k	cs 103k
en→x											
Full FT	610m	17.8	32.9	33.1	23.5	10.1	42.7	43.5	18.7	35.2	25.2
Adapter (b=1024)	50m	18.0	33.3	32.8	22.9	9.9	37.9	42.8	18.2	34.6	24.3
X-attention	50m	16.8	27.7	30.3	21.2	8.8	39.5	40.8	16.3	33.5	22.2
BitFit	335k	12.8	22.7	23.3	16.6	5.3	30.9	30.9	9.5	26.8	15.6
Adapter (b=5)	320k	13.7	22.7	23.9	15.4	5.8	29.3	32.3	9.9	27.3	15.4
From Scratch	40m	25.0	23.9	22.9	15.3	5.5	32.5	35.4	11.0	26.2	17.0
x→en											
Full FT	610m	33.9	38.2	34.1	29.6	23.5	44.8	49.4	31.7	36.0	34.3
Adapter (b=1024)	50m	32.8	38.0	33.5	28.9	22.9	44.4	48.6	30.6	35.2	32.9
X-attention	50m	30.8	32.9	31.6	26.8	19.7	41.9	43.8	27.0	34.0	31.1
BitFit	335k	23.6	29.5	25.9	22.0	14.9	35.2	38.8	21.1	28.1	26.0
Adapter (b=5)	320k	23.2	29.9	25.7	21.8	15.4	34.8	38.2	21.0	27.4	26.4
From Scratch	40m	20.9	27.3	26.3	19.2	11.6	34.3	39.4	19.1	21.9	23.8

Table 9: x↔en results in terms BLEU for mBART experiments.

	M2M-100						**mBART**					
	en <	> fi	en <	> et	fi <	> et	en <	> fi	en <	> et	fi <	> et
Full FT	43.9	37.9	40.4	33.4	33.6	33.4	45.4	39.8	42.3	35.5	34.8	35.4
Adapter (b=1024)	42.7	35.5	39.6	30.9	31.6	31.5	45.3	39.1	41.9	33.8	33.6	33.8
X-attention	42.9	35.9	39.5	31.2	31.6	31.1	40.6	34.2	36.1	28.9	28.5	29.1
BitFit	35.4	25.6	33.9	22.9	26.8	26.1	28.9	18.9	25.0	13.8	18.0	17.3
Adapter (b=5)	36.1	26.8	34.3	23.2	26.9	26.5	28.9	19.2	24.3	14.6	18.2	16.9
Adapter (b=2; 1.2B)	41.9	32.0	39.6	28.8	31.8	31.4	-	-	-	-	-	-
No FT (1.2B)	40.3	28.6	38.1	27.3	31.3	31.0	-	-	-	-	-	-
No FT (484M)	34.1	23.6	32.9	22.6	26.8	26.2	-	-	-	-	-	-

Table 10: (en, et, fi) results in terms of BLEU for M2M-100 and mBART experiments. Note that BLEU scores are not directly comparable as the datasets are different for each language-pair. For a comparison between fine-tuning methods, we refer to relative performances over full fine-tuning (Fig. 4).

6.2 Contribution and impact

There is a little public work building on the paper, probably due to this paper being put online fairly recently. We hope this work can provide a strong baseline/starting point for future research combining PEFT methods and machine translation. In particular it seems important to test on a diverse array of language pairs and dataset sizes since performance varies significantly when those factors are changed.

We cover some limitations of this work in section 8 of the paper, e.g. we do not cover all parameter efficient fine-tuning methods, and do not include analysis of the largest models available, such as the 11 billion parameter M2M or mT5 (Xue et al., 2021) models similar to the encoder-decoder models we use in this paper, or much larger autoregressive language models (Chowdhery et al., 2022).

Future directions for this line of work might include the following: we could have a greater focus on memory efficiency as opposed to parameter-efficiency. For example methods such as a ladder side-tuning (Sung et al., 2022) which don't change any parameters 'inside' the pre-trained model. They instead build a small network alongside 'frozen' the pre-trained model, initializing its weights from a sparse version of the pre-trained model and taking input from pre-trained model hidden states. This means we don't have to backpropagate through any of the pre-trained model, but still have an expressive architecture. A simpler alternative we covered in Chapter 4 is to only fine-tune the last few layers of the pre-trained model. This way we can stop back-propagation after only a few layers. Such techniques might be more important as model scale increases.

Machine translation has so far been left out from many advances in zero-shot and few-shot learning brought about by scaling up transformer language models, where a common paradigm is to 'teach' the model how to perform some tasks by including examples or instructions in the context window or 'prompt' of the language model. We can improve the ability of models to perform tasks in a zero-shot manner by fine-tuning language models on large numbers of tasks (e.g. many QA, NLI or commonsense reasoning datasets) (Sanh et al., 2022; Wei et al., 2022). And the largest models available can perform machine translation without any examples (Chowdhery et al., 2022). However they were tested for translation pairs such as English and German where huge amounts of training data are available and zero-shot performance is perhaps not particularly helpful.

It would be instructive to see how such models perform on lower-resource language pairs or unusual domains, and how zero-shot performance compares to fine-tuning with a small amount of data as we explored in the previous paper. Our experiments in multilingual, multi-domain fine-tuning from section 5.4 would be interesting to scale up in both model size and number of domains, in light of the promising results for fine-tuning on diverse English tasks for the largest pre-trained language models.

Our results also don't currently show significant reduction in the importance of language pair distance for the performance of PEFT methods, or zero-shot translation performance for M2M, as we scale up model size. The simplest solution to this problem is perhaps just to scale up models and data, since absolute performance goes up even if performance relative to similar language pairs doesn't. Other potential solutions might include over-sampling distant language pairs during pre-training, or perhaps modularity in the form of language-specific adapters during pre-training, as done by Pfeiffer et al. (2022).

Chapter 7

Conclusion

Pre-trained language models are powerful tools, and they have the ability to improve performance on a variety of NLP tasks. They offer the promise of no longer needing separate models for separate tasks, instead using a single model with knowledge of many tasks. This allows us to simplify the deployment of these models to production, and is a step towards the goal of 'general intelligence'. However, the strongest performance still comes by updating the parameters of (i.e. fine-tuning) a pre-trained model. But with model sizes up to the hundreds of billions of parameters (Chowdhery et al., 2022), even small scale fine-tuning is out of reach for users with fewer resources. This thesis presented work that introduced techniques to reduce the cost of fine-tuning pre-trained models, in terms of memory, space and computation required. These techniques are known as 'parameter-efficient fine-tuning'.

In Chapter 3 we presented some initial work on improving multi-task fine-tuning on a relatively small-scale model (BERT) by adding task-specific modules on top of a pre-trained model. In Chapter 4 we presented a survey of different parameter-efficient fine-tuning architecture choices, and found these techniques scaled gracefully to much larger pre-trained models. These chapters focused on English Classification and Natural Language Inference tasks. We also presented work on the more challenging task of machine translation with multilingual pre-trained models. In Chapter 5 we discussed the importance of tuning encoder-decoder attention parameters to machine translation performance, and the superiority of starting with a multilingual as opposed to English-only pre-trained model.

We also presented another application of parameter-efficient methods, being able to combine language-specific and domain-specific modules in a zero-shot fashion in order to achieve cross-lingual transfer for a particular domain such as legal text. Finally in Chapter 6, we presented a large empirical study of which factors allow parameter-efficient techniques to work for machine translation, finding for example that these techniques work best when fine-tuning on language pairs where the source and target language are not distantly related.

The thesis so far has focused on either encoder-decoder or encoder-only pre-trained models. These models are strongest when fine-tuned for some task of interest (Tay et al., 2022b). However a technique showing a lot of promise currently is taking a decoder-only language model trained on just autoregressive language modeling and 'teaching' the model how to perform some task by including examples or instructions in natural language in the context window or 'prompt' of the language model (Brown et al., 2020; Chowdhery et al., 2022; Radford et al., 2019), for example setting the prompt as 'Generate a list of questions: 1) What's your favourite film? 2) Where were you born? 3)' and sampling from the model will generate a list of questions. This is typically known as 'in-context learning'. Benefits of this approach include being able to re-use

the same model for any task that can be expressed in natural language and can fit inside the context window, being able to critique (Saunders et al., 2022) or edit model output and generate new output taking into account this feedback, and being able to specify step-by-step reasoning in the prompt to improve performance on tasks that require it.

Where do parameter-efficient techniques fit in with this paradigm? One simple answer is that it may be difficult to include all the information needed to successfully obtain some capability or solve some task in the language model prompt. Longer prompts mean more computation in each forward pass of the model, and in particular the computational cost of the transformer attention mechanism scales quadratically (although various techniques can allow more graceful scaling with length, e.g. Tay et al. (2022a)) with sequence length. If we know that a particular prompt will not change, we could 'distill' it, by training a language model to mimic the output of a prompted model, e.g. we could minimize the KL-divergence between $p(\vec{x}|\vec{c})$ and $p(\vec{x})$ for a particular prompt $\vec{c}$. If c was the question-generation prompt from the previous paragraph, this process would create a model which always just generated a list of questions, without any prompting. We could distill the prompted model into a copy of the language model weights by updating every weight as done by Askell et al. (2021).

But parameter-efficient techniques seem a more natural fit for the following reasons: we retain some of the flexibility of the original in-context learning setting since fine-tuning and storage costs are lower, allowing for more experimentation, and parameter-efficient techniques seem to work better in lower data regimes (see e.g. Chapter 5, section 6.1), which we might want to avoid expensive large-scale fine-tuning. Alternatively we could train an adapter or similar architecture to distil a variety of different prompts, taking the prompt as input, as well as the standard pre-trained model hidden state. This way we could retain the flexibility of having various prompts (and potentially generalizing to variations not in the training set), but avoid the cost of including them in the standard transformer attention mechanism etc.

Although in-context learning is promising, decoder-only models still benefit from fine-tuning (as do encoder-decoder or encoder-only models). Examples include fine-tuning on human preferences (Stiennon et al., 2020) (either with reinforcement learning or supervised learning) or fine-tuning on a specific domain such as mathematical or scientific texts (Lewkowycz et al., 2022). There are many 'skills' that it would be useful for these models to have: better performance on low-resource languages, the ability to browse the web (Nakano et al., 2021), use a calculator or execute code, retrieval from a database of relevant documents (Borgeaud et al., 2021), generate a particular output style like informal language, only provide answers to queries for which the model is sufficiently confident, and so on. While we could fine-tune the model separately on each of these, or all of them in a multi-task fashion, one useful capability would be the ability to 'mix-and-match' these skills.

Building on Pfeiffer et al. (2020), who trained independent language and 'task' adapters which could be composed in a zero-shot fashion and our work in Chapter 5 where we similarly composed domain and language adapters for domain adaptation in machine translation, we can imagine a set up where we compose various skills. For example language and code adapters to allow coding ability in languages other than the typical English, or adapters trained for retrieval and ones trained on medical data in order to retrieve relevant medical documents, and so on.

This direction fits under the broad concept of 'modular' deep learning. Modularity focuses on the development of specialized, autonomous modules within a neural network. Each module is designed to handle a distinct function or task, and the system uses conditional routing to direct data to the appropriate subset of modules based on the task at hand. This potentially means we can combine modules to handle new tasks composed of previously learned skills. Parameter-efficient fine-tuning, as discussed in the thesis, seeks to minimize the computational and memory costs of adapting pre-trained models to new tasks. This approach employs techniques like adapter layers to update task-specific subset of the model's parameters. This fits within the broader scope

of modularity, as these techniques can be thought of as a sort of modular update mechanism allowing for targeted updates without retraining the entire network.

Modularity and parameter-efficient fine-tuning intersect in their underlying motivation to create more versatile, efficient, and specialized models. However, modularity, as it is more broadly conceived, focuses on conditional routing and the assembly of multiple specialized modules. In contrast, parameter-efficient fine-tuning focuses on identifying and optimizing a minimal number of parameters that need to be adapted for a new task, often within the bounds of an existing model architecture. Both approaches share the vision of a mix-and-match capability, where various specialized skills can be combined to enable enhanced performance across a range of tasks without the need for redundant or extensive retraining.

Parameter-efficient techniques might be useful for understanding exactly how fine-tuning is changing a particular pre-trained model. A large body of research seeks to understand how pre-trained models work. For example, finding our which layers encode information about concepts like syntax or semantics (Tenney et al., 2019), finding particular neural 'circuits' (subsets of the entire model that have an interpretable function) that are useful for in-context learning (Elhage et al., 2021; Olsson et al., 2022), or finding MLP layer neurons that only activate for certain human-understandable concepts (Elhage et al., 2022). Since parameter-efficient methods are added 'on top' of the pre-trained model, they don't offer any immediate benefit for better understanding the base model.

However we may be interested in how a fine-tuning stage *changes* the base model. Since full fine-tuning modifies every parameter, understanding exactly what changes were made to the model internals might end up being as challenging a task as understanding the base model in the first place. Additionally, assuming we already have a decent understanding of particular parts of the base model, a naive fine-tuning approach could modify those parts, meaning we have to start from scratch. Freezing the base model and only fine-tuning small modules might offer a solution to these problems, since we cannot 'overwrite' any existing components of the base model, and we have fewer parameters to analyze (Although the fine-tuned parameters could interact in complex ways with the base model parameters).

In Chapter 4 we saw how simple parameter-efficient transfer learning architectures worked well even as model size scaled up. This simplicity might be an advantage for the purposes of interpretability. For example we can set the bottleneck dimension of adapter modules to be very small, leaving us with a small number of neurons to interpret. This setting could also motivate new architectures. If we wanted to examine the effects of fine-tuning on particular attention heads, we might modify the 'LoRA' architecture (which consists of only training low-rank 'corrections' to transformer weight matrices; see Chapter 4). LoRA adds a low-rank correction to the concatenated form of the query and value matrices ($\mathbf{W}_q, \mathbf{W}_v$ in Chapter 1, section 2.3.1) rather than the query and value matrices specific to each head ($\mathbf{W}_q^{(e)}, \mathbf{W}_v^{(e)}$ in Chapter 1, section 2.3.1). Adding a low-rank correction to each head independently (e.g. to each $\mathbf{W}_q^{(e)}$ etc.) might be more useful for interpretability.

Finally, in the previous chapter we mentioned some methods that focus much more on reducing memory usage rather than parameter count: ladder side-tuning (Sung et al., 2022) which doesn't change any parameters 'inside' the pre-trained model, instead fine-tuning a new network alongside the pre-trained model, and a simpler alternative we covered in Chapter 4 where we only fine-tune the last few layers of the pre-trained model. Although modifying the internals of the pre-trained model like we do in most of this thesis provides a performance boost compared to fine-tuning the last few layers, it may turn out that as model scale increases these differences go away, and in that case increasing attention should be put on the best way to do this. For example, do we have to fine-tune the entirety of the last few layers or can we tune a subset, do we need extra connections from previous layers as in Sung et al. (2022), can we recreate capabilities like composing language

and task adapters (Pfeiffer et al., 2020) in this regime?

Overall, while parameter-efficient techniques may have to adapt to an era of larger and/or decoder-only models, they seem likely to remain a useful tool in the toolkit, as long as large-scale pre-trained models are popular.

Bibliography

N. Arivazhagan, A. Bapna, O. Firat, D. Lepikhin, M. Johnson, M. Krikun, M. X. Chen, Y. Cao, G. Foster, C. Cherry, W. Macherey, Z. Chen, and Y. Wu. Massively multilingual neural machine translation in the wild: Findings and challenges, 2019.

A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, J. Kernion, K. Ndousse, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, and J. Kaplan. A general language assistant as a laboratory for alignment, 2021. URL `https://arxiv.org/abs/2112.00861`.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

A. Baevski, H. Zhou, A. Mohamed, and M. Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020. URL `https://arxiv.org/abs/2006.11477`.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL `http://arxiv.org/abs/1409.0473`.

E. Ben Zaken, Y. Goldberg, and S. Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL `https://aclanthology.org/2022.acl-short.1`.

S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. v. d. Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. d. L. Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. W. Rae, E. Elsen, and L. Sifre. Improving language models by retrieving from trillions of tokens, 2021. URL `https://arxiv.org/abs/2112.04426`.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html`.

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

S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

L. Chrisman. Learning recursive distributed representations for holistic computation. *Connection Science*, 3(4):345–366, 1991. doi: 10.1080/09540099108946592. URL https://doi.org/10.1080/09540099108946592.

A. Chronopoulou, D. Stojanovski, and A. Fraser. Language-family adapters for multilingual neural machine translation, 2022. URL https://arxiv.org/abs/2209.15236.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL http://arxiv.org/abs/1412.3555. cite arxiv:1412.3555Comment: Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, 2019.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020. URL https://openreview.net/pdf?id=r1xMH1BtvB.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch, 2011. URL https://arxiv.org/abs/1103.0398.

A. Cooper Stickland, A. Berard, and V. Nikoulina. Multilingual domain adaptation for NMT: Decoupling language and domain information with adapters. In *Proceedings of the Sixth Conference on Machine Translation*, pages 578–598, Online, Nov. 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.wmt-1.64.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, June 2019. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H.-T. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models, 2022.

N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

N. Elhage, T. Hume, C. Olsson, N. Nanda, T. Henighan, S. Johnston, S. ElShowk, N. Joseph, N. DasSarma, B. Mann, D. Hernandez, A. Askell, K. Ndousse, Jones, , D. Drain, A. Chen, Y. Bai, D. Ganguli, L. Lovitt, Z. Hatfield-Dodds, J. Kernion, T. Conerly, S. Kravec, S. Fort, S. Kadavath, J. Jacobson, E. Tran-Johnson, J. Kaplan, J. Clark, T. Brown, S. McCandlish, D. Amodei, and C. Olah. Softmax linear units. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/solu/index.html.

J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

M. Esplà, M. Forcada, G. Ramírez-Sánchez, and H. Hoang. ParaCrawl: Web-scale parallel corpora for the languages of the EU. In *Proceedings of Machine Translation Summit XVII: Translator, Project and User Tracks*, pages 118–119, Dublin, Ireland, Aug. 2019. European Association for Machine Translation. URL `https://aclanthology.org/W19-6721`.

A. Fan, S. Bhosale, H. Schwenk, Z. Ma, A. El-Kishky, S. Goyal, M. Baines, O. Celebi, G. Wenzek, V. Chaudhary, N. Goyal, T. Birch, V. Liptchinsky, S. Edunov, E. Grave, M. Auli, and A. Joulin. Beyond english-centric multilingual machine translation. *arXiv preprint*, 2020.

B. J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambridge, MA, USA, 1998. ISBN 026206202X.

P. Gage. A new algorithm for data compression. 1994. URL `http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM`.

M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL `https://aclanthology.org/2021.emnlp-main.446`.

M. Gheini, X. Ren, and J. May. Cross-attention is all you need: Adapting pretrained Transformers for machine translation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1765, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.132. URL `https://aclanthology.org/2021.emnlp-main.132`.

D. Guo, A. Rush, and Y. Kim. Parameter-efficient transfer learning with diff pruning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4884–4896, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.378. URL `https://aclanthology.org/2021.acl-long.378`.

Z. S. Harris. Distributional structure. *¡i¿WORD¡/i¿*, 10(2-3):146–162, 1954. doi: 10.1080/00437956.1954.11659520. URL `https://doi.org/10.1080/00437956.1954.11659520`.

J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=0RDcd5Axok`.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. URL `https://ieeexplore.ieee.org/abstract/document/6795963/`.

N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799, 2019. URL `https://openreview.net/forum?id=H1NjknZO-H`.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *ArXiv preprint*, abs/2106.09685, 2021. URL https://arxiv.org/abs/2106.09685.

M. Huh, P. Agrawal, and A. A. Efros. What makes imagenet good for transfer learning?, 2016. URL https://arxiv.org/abs/1608.08614.

D. Jurafsky and J. H. Martin. *Speech and Language Processing (3rd Edition Draft)*. USA, 2023. URL https://web.stanford.edu/~jurafsky/slp3/.

N. Kalchbrenner and P. Blunsom. Recurrent convolutional neural networks for discourse compositionality, 2013. URL https://arxiv.org/abs/1306.3584.

R. Karimi Mahabadi, S. Ruder, M. Dehghani, and J. Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 565–576, Online, Aug. 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.47. URL https://aclanthology.org/2021.acl-long.47.

R. Karimi Mahabadi, S. Ruder, M. Dehghani, and J. Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics*, 2021b.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

T. Kudo and J. Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://www.aclweb.org/anthology/D18-2012.

H. Lai, A. Toral, and M. Nissim. Multilingual pre-training with language and task adaptation for multilingual text style transfer, 2022. URL https://arxiv.org/abs/2203.08552.

H. Le, J. M. Pino, C. Wang, J. Gu, D. Schwab, and L. Besacier. Lightweight adapter tuning for multilingual speech translation. *CoRR*, abs/2106.01463, 2021. URL https://arxiv.org/abs/2106.01463.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. *ArXiv preprint*, abs/2104.08691, 2021. URL https://arxiv.org/abs/2104.08691.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703.

A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari, and V. Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ArXiv preprint*, abs/2107.13586, 2021a. URL https://arxiv.org/abs/2107.13586.

X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021b. URL https://arxiv.org/pdf/2103.10385.pdf.

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

Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020. doi: 10.1162/tacl_a_00343. URL https://aclanthology.org/2020.tacl-1.47.

D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999. doi: 10.1109/ICCV.1999.790410.

R. K. Mahabadi, J. Henderson, and S. Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *ArXiv preprint*, abs/2106.04647, 2021. URL https://arxiv.org/abs/2106.04647.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

K. P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL probml.ai.

R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

G. Neubig. Neural machine translation and sequence-to-sequence models: A tutorial. *CoRR*, abs/1703.01619, 2017. URL http://arxiv.org/abs/1703.01619.

C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

J. Pfeiffer, I. Vulić, I. Gurevych, and S. Ruder. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.617. URL https://aclanthology.org/2020.emnlp-main.617.

J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online, 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.eacl-main.39.

J. Pfeiffer, N. Goyal, X. Lin, X. Li, J. Cross, S. Riedel, and M. Artetxe. Lifting the curse of multilinguality by pre-training modular transformers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3479–3495, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.255. URL https://aclanthology.org/2022.naacl-main.255.

J. Philip, A. Berard, M. Gallé, and L. Besacier. Monolingual adapters for zero-shot neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4465–4470, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.361. URL https://aclanthology.org/2020.emnlp-main.361.

M. T. Pilehvar and J. Camacho-Collados. WiC: The word-in-context dataset for evaluating context-sensitive meaning representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, 2019. URL https://arxiv.org/abs/1808.09121.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, E. Rutherford, T. Hennigan, J. Menick, A. Cassirer, R. Powell, G. v. d. Driessche, L. A. Hendricks, M. Rauh, P.-S. Huang, A. Glaese, J. Welbl, S. Dathathri, S. Huang, J. Uesato, J. Mellor, I. Higgins, A. Creswell, N. McAleese, A. Wu, E. Elsen, S. Jayakumar, E. Buchatskaya, D. Budden, E. Sutherland, K. Simonyan, M. Paganini, L. Sifre, L. Martens, X. L. Li, A. Kuncoro, A. Nematzadeh, E. Gribovskaya, D. Donato, A. Lazaridou, A. Mensch, J.-B. Lespiau, M. Tsimpoukelli, N. Grigorev, D. Fritz, T. Sottiaux, M. Pajarskas, T. Pohlen, Z. Gong, D. Toyama, C. d. M. d'Autume, Y. Li, T. Terzi, V. Mikulik, I. Babuschkin, A. Clark, D. d. L. Casas, A. Guy, C. Jones, J. Bradbury, M. Johnson, B. Hechtman, L. Weidinger, I. Gabriel, W. Isaac, E. Lockhart, S. Osindero, L. Rimell, C. Dyer, O. Vinyals, K. Ayoub, J. Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu, and G. Irving. Scaling language models: Methods, analysis amp; insights from training gopher, 2021. URL https://arxiv.org/abs/2112.11446.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv preprint*, abs/1910.10683, 2019. URL https://arxiv.org/abs/1910.10683.

S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30, 2017.

S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

S. Ruder, J. Pfeiffer, and I. Vulić. Modular and parameter-efficient fine-tuning for NLP models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, pages 23–29, Abu Dubai, UAE, Dec. 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-tutorials.5.

V. Sanh, A. Webson, C. Raffel, S. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, A. Raja, M. Dey, M. S. Bari, C. Xu, U. Thakker, S. S. Sharma, E. Szczechla, T. Kim, G. Chhablani, N. Nayak, D. Datta, J. Chang, M. T.-J. Jiang, H. Wang, M. Manica, S. Shen, Z. X. Yong, H. Pandey, R. Bawden, T. Wang, T. Neeraj, J. Rozen, A. Sharma, A. Santilli, T. Fevry, J. A. Fries, R. Teehan, T. L. Scao, S. Biderman, L. Gao, T. Wolf, and A. M. Rush. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9Vrb9D0WI4.

W. Saunders, C. Yeh, J. Wu, S. Bills, L. Ouyang, J. Ward, and J. Leike. Self-critiquing models for assisting human evaluators, 2022. URL https://arxiv.org/abs/2206.05802.

R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1009. URL https://www.aclweb.org/anthology/P16-1009.

A. C. Stickland and I. Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995, 2019. URL http://proceedings.mlr.press/v97/stickland19a.html.

A. C. Stickland, A. Bérard, and V. Nikoulina. Multilingual domain adaptation for NMT: decoupling language and domain information with adapters. *Sixth Conference on Machine Translation (WMT2021)*, 2021a. URL https://www.statmt.org/wmt21/pdf/2021.wmt-1.64.pdf.

A. C. Stickland, X. Li, and M. Ghazvininejad. Recipes for adapting pre-trained monolingual and multilingual models to machine translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3440–3453, Online, Apr. 2021b. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2021.eacl-main.301.

N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano. Learning to summarize with human feedback. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf.

Y.-L. Sung, J. Cho, and M. Bansal. Lst: Ladder side-tuning for parameter and memory efficient transfer learning, 2022. URL https://arxiv.org/abs/2206.06522.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.

P. Swietojanski, J. Li, and S. Renals. Learning hidden unit contributions for unsupervised acoustic model adaptation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24 (8):1450–1463, 2016. doi: 10.1109/TASLP.2016.2560534.

Y. Tay, Z. Zhao, D. Bahri, D. Metzler, and D.-C. Juan. Hypergrid: Efficient multi-task transformers with grid-wise decomposable hyper projections, 2020. URL https://arxiv.org/abs/2007.05891.

Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey, 2022a.

Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, J. Wei, X. Wang, H. W. Chung, D. Bahri, T. Schuster, H. S. Zheng, D. Zhou, N. Houlsby, and D. Metzler. Ul2: Unifying language learning paradigms, 2022b. URL https://arxiv.org/abs/2205.05131.

I. Tenney, D. Das, and E. Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1452. URL https://aclanthology.org/P19-1452.

C. Tran, C. Wang, Y. Tang, Y. Tang, J. M. Pino, and X. Li. Cross-modal transfer learning for multilingual speech-to-text translation. *CoRR*, abs/2010.12829, 2020. URL https://arxiv.org/abs/2010.12829.

A. Üstün and A. Cooper Stickland. When does parameter-efficient transfer learning work for machine translation? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7919–7933, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.540.

A. Üstün, A. Bisazza, G. Bouma, and G. van Noord. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.180. URL https://www.aclweb.org/anthology/2020.emnlp-main.180.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

T.-T. Vu, S. Khadivi, X. He, D. Phung, and G. Haffari. Can domains be transferred across languages in multi-domain multilingual neural machine translation?, 2022. URL https://arxiv.org/abs/2210.11628.

J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann. Bloomberggpt: A large language model for finance, 2023.

L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua, and C. Raffel. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.41. URL https://aclanthology.org/2021.naacl-main.41.

K. Zhang, J. Yu, Z. Yan, Y. Liu, E. Adhikarla, S. Fu, X. Chen, C. Chen, Y. Zhou, X. Li, L. He, B. D. Davison, Q. Li, Y. Chen, H. Liu, and L. Sun. Biomedgpt: A unified and generalist biomedical generative pre-trained transformer for vision, language, and multimodal tasks, 2023.