Adapting Pretrained Models for Machine Translation

Aditya Kurniawan

Supervisor: Dr. Marc Tanti

Faculty of ICT

University of Malta

enter a date

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Faculty of ICT

Declaration

I, the undersigned, declare that the dissertation entitled:

Adapting Pretrained Models for Machine Translation

submitted is my work, except where acknowledged and referenced.

Aditya Kurniawan

enter a date

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisors Prof. Ondrej Bojar and my co-supervisor Dr Marc Tanti for their overall support, thorough feedback, and guidance during the creation of this thesis.

Secondly, I would like to thank the Erasmus Mundus European Master Program in Language and Communication Technologies (LCT) for the scholarship that allowed me to go through this two-year journey.

Thirdly, I thank MetaCentrum for providing the resources required to complete this thesis. We used the computational resources supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Finally, I would also like to thank my family and friends for their support throughout my study.

Abstract

Pre-trained language models received extensive attention in recent years. However, it is still challenging to incorporate a pre-trained model such as BERT into natural language generation tasks. This work investigates a recent method called adapters as an alternative to fine-tuning in machine translation. Adapters are a promising approach that allows fine-tuning only a tiny fraction of a pre-trained network. We show that with proper initialization, adapters can help achieve better performance than training models from scratch while training substantially fewer weights than the original model. We further show that even with randomly set weights used as the base models for fine-tuning, we can achieve similar performance to one of the baseline models, bypassing the need to train hundreds of millions of weights in the pre-training phase. Furthermore, we study the effectiveness of adapters in the Transformer model for machine translation. We put adapters either in the encoder or the decoder only, and we also attempt to down-scale the pre-trained model size to decrease GPU memory demands. We found that incorporating adapters in the encoder alone matches the setup's performance when we include the adapters on both the encoder and decoder. Finally, our down-scaling study found that using only half of the original pre-trained weights can positively impact the performance when fine-tuned with adapters. Our experiments show that we can get almost the same performance as the original BERT model after fine-tuning the cross-attention layer.

Contents

List of Figures

List of Tables

Introduction

Pre-trained language models ([29, 37]) have received considerable attention in recent years. These models are trained on large-scale corpora and then fine-tuned for a particular downstream task. This method allows pre-trained models to perform well across various natural language processing tasks. One of the most successful models is BERT ([29]). BERT has been most extensively used for common natural language understanding (NLU) tasks. It has been shown that BERT can achieve great performance with relatively straightforward fine-tuning, especially for classification-like tasks.

For natural language generation (NLG), incorporating BERT is still challenging. According to [89], simply incorporating BERT into the encoder side of the seq2seq architecture can hurt the performance. On the decoder side, BERT does not quite fit either because the bidirectional nature of the model was significantly different from the objective of the conditional language model (predicting the next word).

Fine-tuning all BERT's parameters is inefficient, given that there are approximately 200 million parameters in a single model of BERT. Naive fine-tuning also often results in catastrophic forgetting, where the models forget the previous knowledge they have acquired while improving on the new domain ([50, 87]). This may explain why it is considered harmful to simply fine-tune an initialized encoder component with BERT. It is also known that fine-tuning large pre-trained language models could result in unstable and fragile performance on small datasets.

Adapter is an alternative approach that allows for fine-tuning a model without altering the original network weights ([36, 3]). By leveraging adapters, one can

reduce the number of parameters updated in fine-tuning and make the process computationally less expensive while achieving similar results. Another useful property of the approach with adapters is that they are more robust against catastrophic forgetting than fine-tuning ([34]).

This work uses BERT and its variants as the base pre-trained models and fine-tune them with adapters. We evaluate the models on machine translation with the following objectives:

- We conduct a study to understand the contribution of good representation in the pre-trained language model when fine-tuning using adapters.
- We conduct a study to evaluate the effectiveness of adapters in the seq2seq framework by putting them only in the encoder or the decoder.
- We experiment with down-scaling the pre-trained model size and try to recover the performance from being comparable to the full-sized model.

Thesis Organization

Chapter 1 discusses the theoretical background of machine translation, transfer learning, and a brief overview of the current state of using adapters in various setups.

Chapter 2 reviews the previous related work on transfer learning from models that were pre-trained on language model objectives and the usage of adapters in various disciplines within text and speech domains.

Chapter 3 describes the dataset that we use to train language models and machine translation. We then explain the pre-processing of the dataset and the tokenization to construct the vocabularies. Finally, we describe the framework for the experiments and the automatic evaluation metric.

Chapter 4 presents our attempt to use adapters in machine translation setup. This chapter focuses on the contribution of pre-trained representations when fine-tuning with adapters. We discuss the result of our experiments by referring to the automatic evaluation metric as well as providing our own manual analysis. Furthermore, we discuss the limitation of incorporating BERT in machine translation by providing the translation output errors.

Chapter 5 presents our attempt to understand the effectiveness of adapters and the impact of the pre-trained weights for adapters by placing them only in the encoder or decoder. We then continue the experiments by down-scaling BERT to half of the size and trying to recover the adapter's performance so that it is comparable to the full-sized model. We provide discussions of the phenomenon that happens when reducing the size of the original BERT model.

Conclusion summarizes our findings from the experiments.

1. Background and Motivation

In this chapter, we summarize the background and motivation underlying this work. In Section 1.1, we briefly review the approaches to machine translation (MT) and the comparison between them. Section 1.2 reviews the formulation of neural machine translation as well as the frameworks utilized to solve the problem. Section 1.3 discusses transfer learning (TL) and its variants. Finally, Section 1.4 discusses the adapter module as an alternative methodology for fine-tuning the model and provides reasons why we focus our research on using adapters.

1.1 Machine Translation

On a fundamental level, MT performs the substitution of words from one language to another. However, producing a good translation based on the substitution alone is challenging. It requires understanding the whole sentence that includes phrases and surrounding words in the target language. The problem is exacerbated as words may have more than one meaning, and it is sometimes challenging to determine the exact relation between expressions across languages.

The three most commonly used approaches in MT are rule-based, statistical (SMT) and neural (NMT). Due to the significant effort in manually collecting good dictionary and grammatical rules, an automatic approach such as SMT or NMT seems more appealing. Prior to NMT, a variant of SMT, namely phrase-based

machine translation (PBMT), had been the state-of-the-art for German-to-English language pairs. [11] also shows that PBMT performs well in different language pairs. [41] introduced the first end-to-end neural network for machine translation with an encoder-decoder structure. Their approach encodes a given source text into a continuous vector and further transforms the vector into the target language.

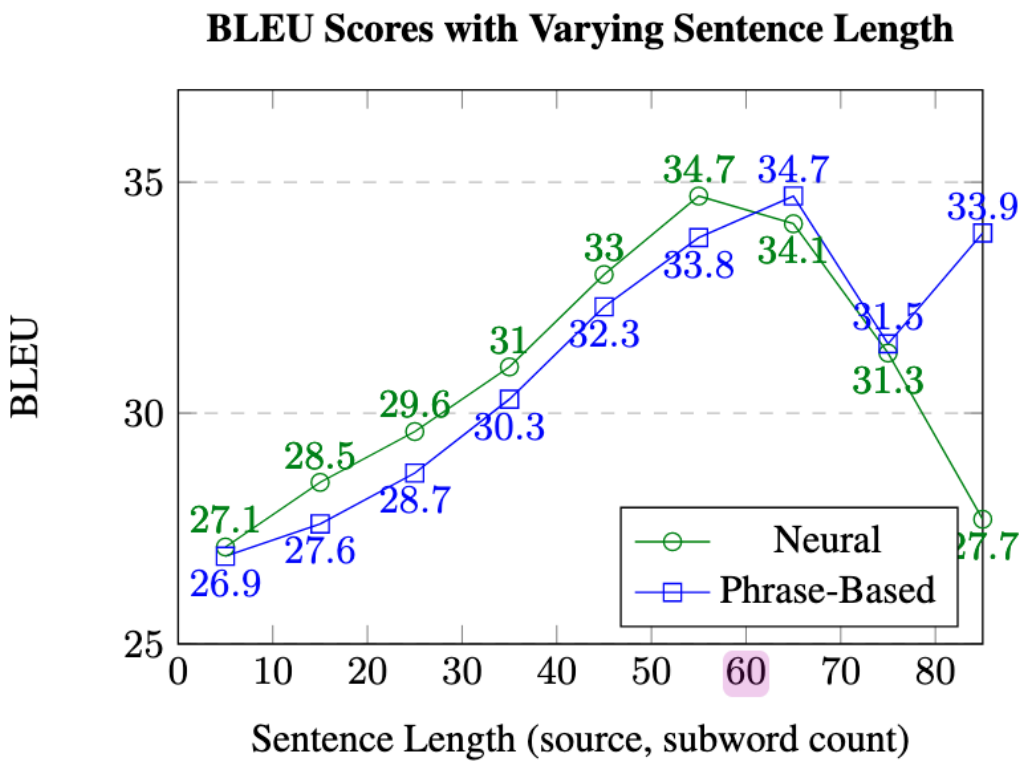

Figure 1.1: Performance of SMT vs NMT on various sentence lengths from [42].

While NMT has been the primary technique used in various MT challenges, such as WMT, there are some advantages and disadvantages. According to [42], NMT suffers from the following phenomena:

- In an out-of-domain scenario, NMT systems suffer more from a quality loss than PBMT. The authors found that the model tends to sacrifice adequacy over fluency completely.
- NMT requires a large amount of data. It is problematic when low-resource

languages are involved in the evaluation.

- NMT still shows weakness in the translation quality for low-frequency words despite its better performance. A similar problem also occurs in PBMT, where the model's performance suffers when low-frequency words occur in the sentence. The problem is especially apparent when the word is entirely unknown.

- Difficulties in translating long sentences. NMT can perform well in short sentences up to 60 words. However, longer sentences (80 and more tokens) show a lower quality of translation. Figure 1.1 shows the performance of SMT and NMT on different sentence lengths.

- Model interpretability. As opposed to PBMT, it is not easy to interpret the behaviour of NMT due to the complexity of the model and its high number of parameters. Furthermore, the training of NMT is also non-deterministic due to random parameter initialization.

Despite its shortcomings, NMT is clearly and by far the current best approach. Among many other advantages, [49] mentioned that the difference is apparent in the output fluency. They mentioned that PBMT models suffer from double negation and translation in morphologically rich languages. These problems cause little to no problem at all in NMT models.

1.2 Neural Machine Translation

We define the translation problem used in this work as a mapping function t from a given source language S and target language T from parallel corpus, that is $t : S \rightarrow T$. A parallel corpus is a set of sentence pairs in two different languages where one sentence in T corresponds to its equivalent in language S. The goal of function t is to find the highest probability of sentence $y \in T$ from $x \in S$, where

$t(x) = argmax_y(p(y|x))$. The probability $p(y|x)$ is the probability estimation given by the NMT model.

The following sections show a quick overview of recent NMT models. They include architecture, advantages, and drawbacks.

1.2.1 RNN Seq2Seq

[76] proposed a sequence-to-sequence (seq2seq) model that employs Recurrent Neural Network (RNN) as the fundamental architecture. The architecture is a straightforward extension of the language model problem. Essentially, the model sequentially predicts the next word given all previous words while incorporating features obtained from the source language.

In MT, the approach is modified by using two similar model architectures for language S and T. For language S we call this component `encoder` and for T we call it `decoder`. The task of the `encoder` is to produce a vector representation of the input sentence from the source language $x \in S$. We define the input sentence as a sequence of tokens from a fixed vocabulary, i.e. the set of tokens permitted in the source language $\{x_1, x_2, ..., x_n\} \in x$. The tokens are then transformed into a vector representation by an embedding matrix. These representations consumed by an RNN result in a new representation that combines features from the embedding and its context. The `decoder` has a similar functionality as the `encoder` where it uses a sequence of tokens from $y \in T$ as its inputs. The `decoder` incorporates additional features from the last vector of the `encoder` and uses it throughout the entire sequence in the target language. For further illustration, we refer to Figure 1.2.

[24] found several disadvantages of using this variant of the seq2seq framework. The models' performance decreased when the length of the source sentence increased. From the previous paragraph, we recall that the `decoder` only uses the last vector of `encoder` as the additional feature. The vector is a fixed-size vector whose size is defined prior to the training. Hence, the vector could be less infor-

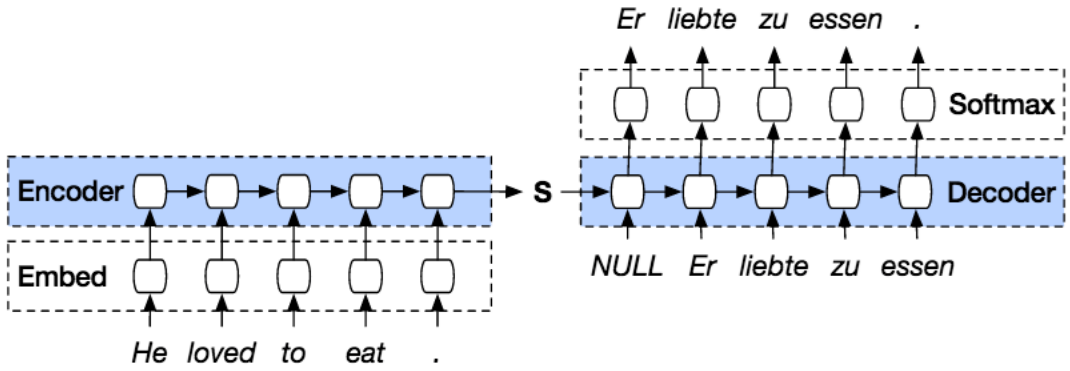

Figure 1.2: Illustration of seq2seq architecture from [40]

mative when the source sentence grows due to the missing information it has to sacrifice during the encoding. In essence, this framework forces the `encoder` to create a good representation by combining the features from all the words in the source sentence within a single vector in a limited size.

Furthermore, RNNs also suffer from another problem where the gradients can be extremely small or large. These problems are often mentioned as vanishing and exploding gradient problems. When the model's gradients are extremely small, RNNs can not learn from the data effectively and struggle, especially with long-range dependencies. On the other hand, when the gradients are extremely large, it can affect the model's parameters by moving them far away from the optimal space. This would disrupt the learning process and cause the model to fail to learn. This problem can happen in seq2seq architecture as we backpropagate the gradients from the last output of the `decoder` to the first input of the `encoder`.

1.2.2 Seq2Seq with Attention

The encoder-decoder model can be extended by jointly learning to translate and align words in the source language to the target language ([1]). This method learns to identify the most relevant sources of information in a source sentence and uses the encoder state vectors associated with the source positions to predict a target word.

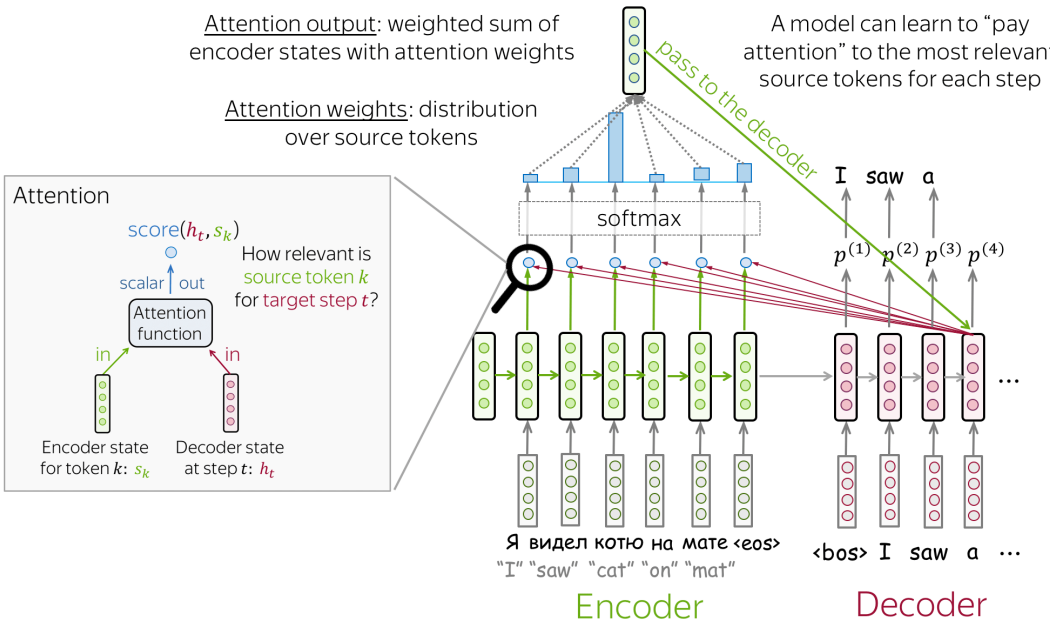

Figure 1.3: Illustration of seq2seq architecture with attention from [79].

This method eliminates the need for a neural model to learn to squash the whole source sentence into a fixed-length vector as proposed by [76]. Instead, it is softly combining vectors from the source sentence that is deemed to contain the most relevant information to be used while decoding each word of the translated message. This allows the model to provide better predictions in long sentences. We can see the illustration of this architecture from Figure 1.3.

The proposed approach achieves significantly better translation quality than the original encoder-decoder model. The improvement is especially evident in longer sentences. The model shows comparable performance to or close to the phrase-based system in the En-Fr pair.

The motivation for this work is to identify the association between the decoder state and the input word. This attention method aims to measure the impact of word representation in the source sentence by looking at the strength of this association to produce the subsequent output word.

1.2.3 Transformer

Recurrent models such as RNNs usually compute each token symbol of the input and produce output sequentially. During these sequential computations, they generate hidden states h_t as a representation of the current position t. These hidden states take into consideration a combination of current input and the previous hidden state h_{t-1}. This behaviour implicitly forces the network to behave in a sequential manner and prevents parallelization within the training procedure. This parallelization becomes very important as the length of the input grows. Several works through factorization tricks [44] and conditional computation [74] have achieved notable improvements in reducing the computational time. However, it does not change the fact that the inherent sequential nature of the model remains.

To alleviate the sequential problem, [78] proposed a new architecture called transformer. This new architecture avoids the recurrent nature of RNN entirely and only relies on the attention mechanism to provide dependencies between words in a sentence. It allows more parallelization and reduces significant training time to achieve a new state of the art in several tasks such as machine translation.

We can refer to Figure 1.4 for illustration of the transformer architecture. The architecture again consists of two main components, an encoder and a decoder. The attention on the encoder side assigns an attention score to each word in the source sentence. The authors claim that compared to the sequential models, transformer is able to carry information between any pair of words in a single step and help the model reaches better quality while also improving the training speed. The authors further propose an additional attention layer on the decoder that refers to the representation in the encoder for a better context in a similar way as the attention in RNNs. This is helpful for tasks such as machine translation, where context from the source side is essential for the prediction.

Based on [47], despite its contribution to leading many breakthroughs in natural language processing (NLP) space, transformer requires non-trivial efforts in training the model. Contrary to the other neural layers, such as recurrent neu-

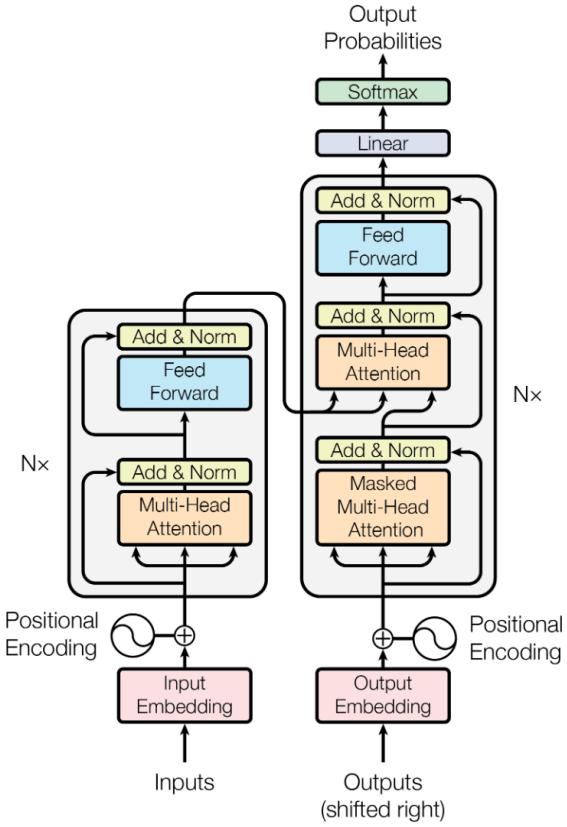

Figure 1.4: Illustration of Transformer model. Figure reprinted from [78].

ral network (RNN) and convolution neural network (CNN), optimization such as stochastic gradient descent (SGD) may converge to bad local optima if not tuned carefully. Furthermore, the warmup stage during training is crucial as it leads to severe consequences such as model divergence when it is omitted during training. We understand from this finding that training transformers and obtaining an optimal performance are not straightforward.

1.3 Transfer Learning

Transfer learning (TL) focuses on transferring knowledge from one problem to a different but related problem. Transfer learning involves a source domain D_S and a source task T_S, as well as a target domain D_T and a target task T_T. We aim

to learn and improve the target conditional distribution $P(Y_T|X_T)$ from D_T by leveraging information from D_S as well as T_S, where $D_S \neq D_T$, and/or $T_S \neq T_T$. We define Y_T as the set of all target labels and X_T as the document or sentence representations. For a better illustration on TL, we can refer to Figure 1.5.

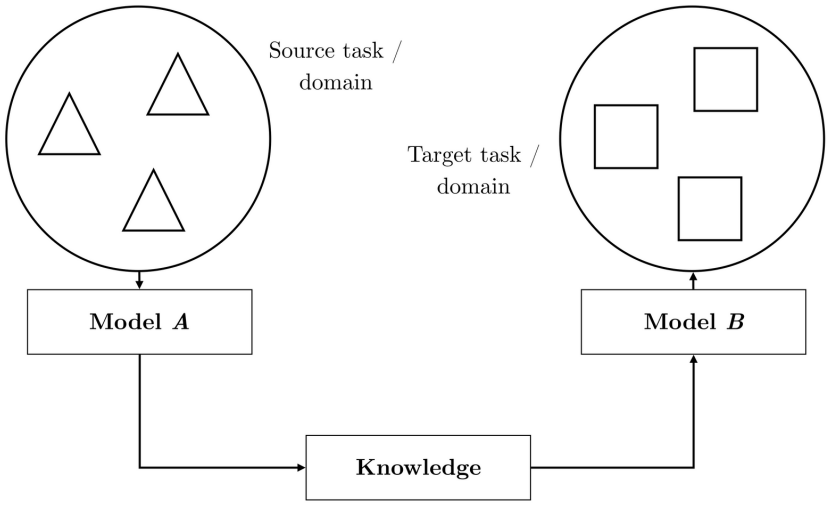

Figure 1.5: An illustration of transfer learning in different domain. Figure reprinted from [68].

[77] describe three ways of how transfer learning can improve performance. Specifically:

- Improving the initial performance at the beginning of training compared to a randomly initialized model when the tasks are similar;
- Shortening the time needed to reach the maximal performance;
- Improving the final performance level compared to training the model without the transfer

To some extent, we can see transfer learning as a way to initialize neural networks with more constraints than the usual definition of weights initialization. In weight initializations, we focus on initializing random weights for any neural network architecture. On the other hand, transfer learning is only applicable to a specific part of the neural network architecture within the same domain problem,

i.e., we can only transfer weights from a network component to another network that uses the same component configuration, such as the number of neurons and layers. Domain problems can be defined as a category of problems such as computer vision, natural language processing, or speech recognition.

For this work, we are specifically interested in conducting domain adaptation by learning the initial knowledge from a dataset with distribution A and later using the knowledge as a new starting point for training in a different domain B. We achieve this by performing a sequential transfer learning technique, where we fine-tune the models that have already trained on distribution A into distribution B. We will discuss the definition of domain adaptation and sequential transfer learning in the following sections to provide more details and examples of both techniques in MT.

1.3.1 Domain Adaptation

In the context of NMT, we can distinguish two categories of domain adaptation. The first category is domain adaptation in the linguistic sense, where we deal with the same language pairs but in different domains. For example, we pre-train a model in WMT data in German→English pair and adapt IWSLT data within the same language. In the second category, we have multilingual adaptation, where we work with entirely different language pairs between the pre-training and the fine-tuning phase. For the scope of this thesis, we limit the problem to domain adaptation and only a single language. We choose not to perform experiments on the multilingual adaptation as we want to isolate our study in understanding our proposed technique in one of the problems in transfer learning.

The goal of domain adaptation is to optimize a model that has been previously trained for a different domain. Models that are optimized on a specific genre (news, speech, medical, literature, and other) have higher accuracy than a system that is optimized for a more generic domain ([32, 35]). This is due to the model's bias over the target domain. When the training data distribution is unbiased towards

the test set in a particular target domain, we expect to reach a similar performance compared to the performance on a set held out from the training data. On the other hand, the performance will decline if the training data distribution is different.

In NMT, adaptation procedure involves training models over two different data distributions ([48, 73, 26]). The models are first trained on an out-of-domain parallel corpus containing broad information. More in-domain training data is introduced to fine-tune the model when the first stage of training (pre-training) has finished. We can see this as a form of transfer learning where the gained knowledge from the out-of-domain corpus is leveraged by the model while fine-tuning in the in-domain corpus.

There are two problems in domain adaptation found by [31]:

- The models are prone to over-fitting when the size of in-domain data is limited.
- The models suffer from catastrophic forgetting when they are fine-tuned. This means that the models' performance in the out-of-domain data will degrade while the performance on in-domain data may be improved.

A proposed solution from [25] addresses all these problems by mixed fine-tuning. Essentially, they combine the out-of-domain corpus and in-domain data when adapting the general model.

1.3.2 Sequential Transfer Learning

Based on [68], there are two existing strategies for applying pre-trained models: feature-based and fine-tuning. The feature-based approach incorporates the pre-trained model outputs as an additional feature to the models in the downstream tasks. The example of this approach can be found in ELMo ([54]). On the other hand, the fine-tuning approach employs the pre-trained model on the same model architecture in the downstream tasks. Several works, such as BERT ([29]) and

OpenAI GPT ([61]), show that fine-tuning provides a significant improvement compared to the feature-based approach. Therefore, we only consider the fine-tuning approach to train our model for this thesis.

Sequential transfer learning is a form of transfer learning that has led to the biggest improvements in NLP so far ([29, 37, 62]). In practice, we aim to perform pre-training to build decent vector representations from a large unlabelled text corpus and then adapt these representations to a target task with labelled data. For a better illustration, we refer to Figure 1.6.

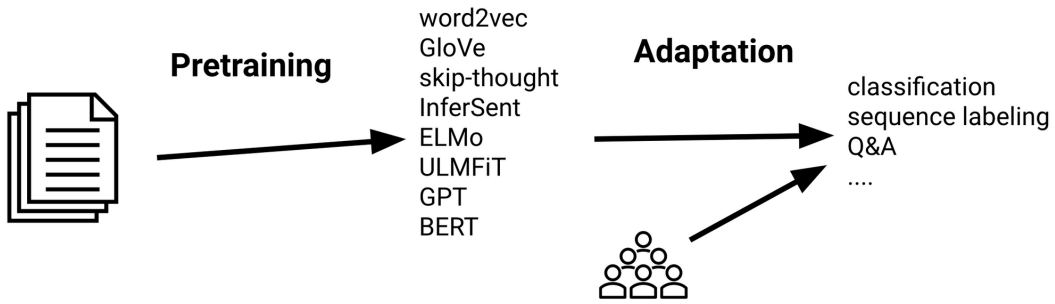

Figure 1.6: Illustration of sequential transfer learning. Figure reprinted from [68].

One of the most prominent examples of sequential transfer learning in NLP is pre-training with a language model objective. Language model pre-training has been shown as an effective objective in improving various NLP tasks ([28, 54, 61, 38]). This includes sentence-level tasks in NLU such as natural language inference ([14, 83]) and sentence paraphrasing [30]. It also has been shown to improve performance in token-level tasks where models are expected to output another token, such as named entity recognition, question answering, and machine translation ([69, 65]). In machine translation, the availability of high-quality parallel data can be a limitation to training good NMT models that can generate good output. Contextual knowledge such as the one from pre-trained models could be a good complement for NMT.

Although the language modelling task looks straightforward from a high-level overview, it is challenging for both machines and humans. For models to provide

a solution, they must understand linguistic phenomena from both syntax and semantics as well as particular knowledge about the world. It has been shown that given enough data, enough computational power, and a large number of parameters, the model can provide a reasonable output ([62]). Several works have shown empirically that language modelling performs better than other pre-training tasks such as translation or autoencoding ([88, 80]).

BERT

This section discusses BERT as one of NLP's most prominent pre-trained models. It was proposed by [29] as they argue that the current language model objective restricts the power of the pre-trained representations, especially when applied for fine-tuning in the downstream tasks. There is a significant limitation of the conditional language models, which only perform unidirectional prediction. For example, we can not simultaneously use RNN in left-to-right and right-to-left (bidirectional RNN) directions, as each direction will provide answers for the other. This restriction can be harmful to sentence-level tasks. Especially in machine translation, it would be more beneficial to encode the sentence in both directions on the encoder part as we are only concerned with obtaining a good representation from the source sentence. Further illustration can be seen in Figure 1.7.

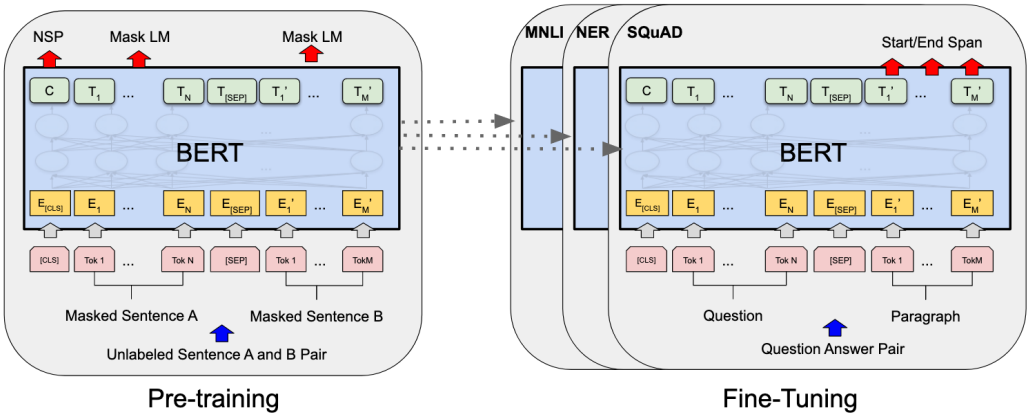

Figure 1.7: Illustration of BERT framework. Figure reprinted from [29].

Despite its success in various NLU tasks, incorporating BERT in NLG remains challenging. [89] found that using pre-trained BERT as the initialization on the encoder side hurts the performance. An explanation for this is that fine-tuning BERT on a complex task requires extra care and may lead to the catastrophic forgetting problem ([50]) of the pre-trained model. On the decoder side, there is a mismatch in initializing the component with BERT due to the conditional nature of the training objective. We understand that we can treat the objective of machine translation in the decoder as a conditional language model. This differs from BERT as it uses a bidirectional objective such as Masked Language Model (MLM). Furthermore, fine-tuning the full weights of BERT is inefficient, considering the enormous number of parameters one needs to modify. It is also tricky to fine-tune BERT with small datasets as the process can be unstable and fragile ([45]).

1.4 Adapters

An adapter is a lightweight layer inserted between the layers of a pre-trained transformer network, and it is fine-tuned on the adaptation corpus. The adapters were proposed by [36] as an alternative to fine-tuning. In this work, we follow the architecture similar to the work of [3] as it is simpler to implement and has been adopted in other works ([58, 67, 56]). Specifically, we follow the adapter implementation of [58] because they have already performed an extensive hyperparameter search to find the best combination of the position and the number of adapters within a single layer, the position of residual connections, the bottleneck residual factors, as well as the non-linearity function within the bottleneck adapter layer within each transformer layer.

Following [58], adapter is defined with the following formulation

$$A_l(h_l, r_l) = U_l(ReLU(D_l(LA_l))) + r_l$$

A_l is the adapter incorporated at layer l, D_l is a down projection layer $D \in R^{h \times d}$

where h is the dimension of the current layer and d is the adapter dimension, U_l is an up projection layer $U \in R^{d \times h}$, and r_l is the residual connection from the previous layer, both h and l are larger than d. We can refer to Figure 1.8 for an illustration of the adapter bottleneck layer and how they are incorporated to the Transformer architecture.

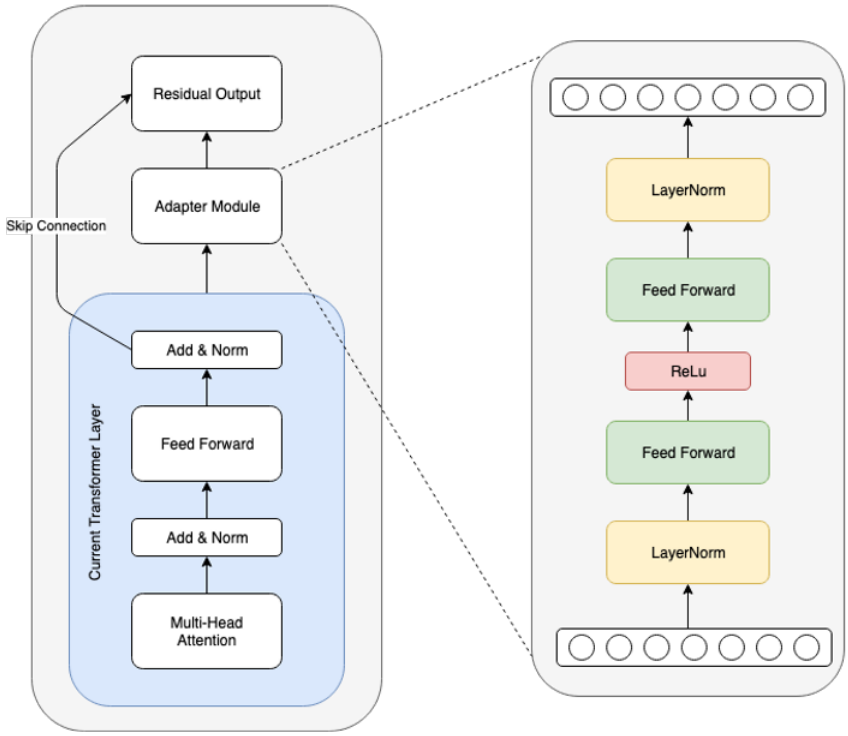

Figure 1.8: Illustration of Adapters.

There are several problems in fine-tuning that adapters are trying to solve:

- The number of parameters in the state-of-the-art NMT has been increasing ([75, 2, 39]), and performing fine-tuning on all parameters is too costly.
- Fine-tuning the whole network weights demands meticulous hyperparameter tuning during the adaptation process, and it is prone to over-fitting ([71, 4]).
- [45] suggests that catastrophic forgetting leads to instability during fine-tuning.

- [51] suggests gradient vanishing also contributes in instability during fine-tuning.
- The high capacity model intensifies the sensitivity to both hyperparameter and over-fitting.

[34] show that fixing the pre-trained layers and only fine-tuning adapters improves the model's stability on various random seeds, enhances adversarial robustness, and leads to a better final performance. We can see Figure 1.9 for the difference between models that were trained with (cluster on the right) and without adapters (cluster on the left) on different pre-training and fine-tuning iterations. The fine-tuned models with adapters show more stability than the ones that were not using adapters. The pre-training models use the same architecture and network configurations during the training but use different iterations to stop the training.

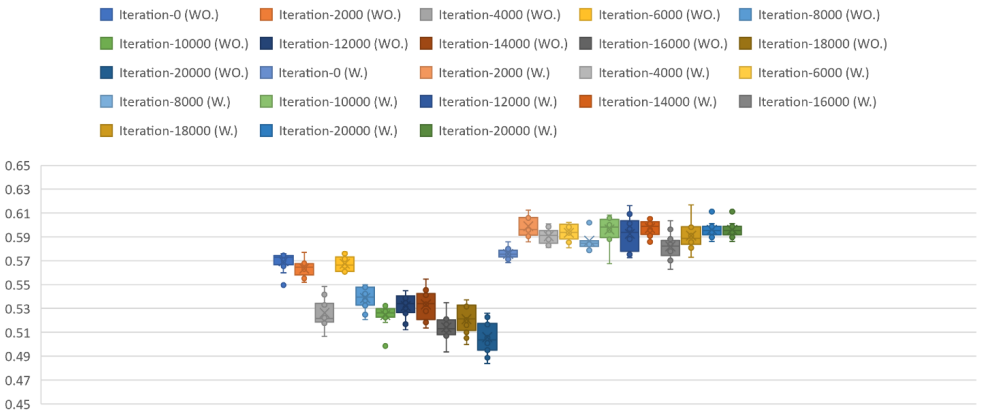

Figure 1.9: Illustration of adapters instability. "W" represents the models that use adapters and "WO" represents the models that do not use adapters. Figure reprinted from [34].

2. Pre-training and Adapters

This chapter reviews the previous work related to this thesis on the adaptation of transformer and BERT with adapters in machine translation. Specifically, in Section 2.1, we review works in incorporating BERT in machine translation. In Section 2.2, we review the usage of adapters using transformer as the base model in various fields such as NLU, automatic speech recognition (ASR), and MT.

2.1 Pre-training Language Models in Machine Translation

This section will discuss three different works that try to incorporate BERT into NMT. The goals of these works are different from the experiments proposed in this thesis because they primarily leverage BERT's feature representation instead of using BERT directly to fine-tune NMT.

[82] propose to acquire knowledge from pre-trained models such as BERT with a framework called A_{PT}. A_{PT} consists of two different modules with different goals. The first goal is to learn a task-specific representation through adaptation from general representation in the pre-trained models and to learn two controlling methods to control the task-specific representation into NMT. They propose to achieve the first goal by using a dynamic fusion mechanism. They claim this method

could provide rich contextual information to model sentences in NMT better. An illustration of their proposed approach can be seen in Figure 2.1. The second goal is to prune the knowledge from pre-trained models via knowledge distillation. They claim this method could help NMT to continuously learn essential knowledge about translation from source sentence to target sentence by using parallel data and help generate better translation by learning from monolingual data. Moreover, based on their empirical results, they achieve the best results when both methods are applied in the encoder and decoder. We refer to Figure 2.2 for illustration of this approach.

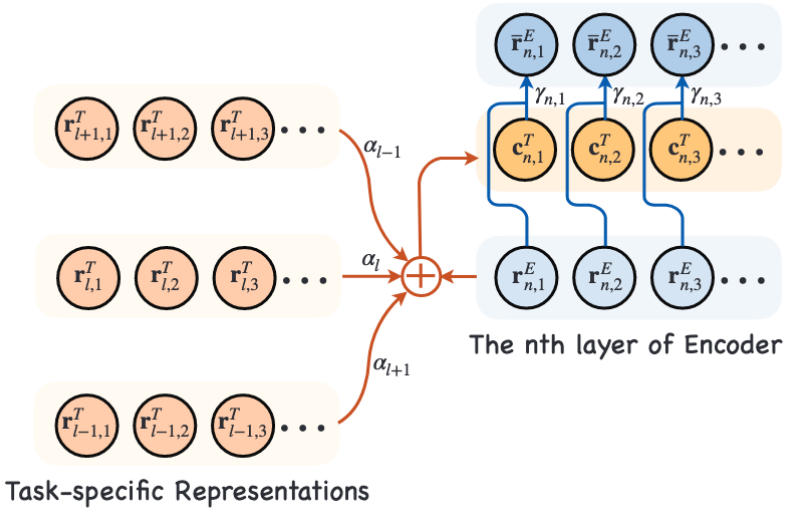

Figure 2.1: Illustration of dynamic fusion mechanism. Figure reprinted from [82].

[86] try to combat issues such as requiring a long time to train NMT models and catastrophic forgetting problem from updating too many of the pre-trained models in the fine-tuning process. They propose a concerted training approach (CT_{NMT}) that consists of three different techniques: 1) Asymptotic distillation; 2) Dynamic switch for knowledge fusion; 3) Rate-scheduled updating. The first technique is used to keep the pre-trained knowledge intact. They achieve this by using the pre-trained BERT as a teacher network and the encoder of the NMT models as the student. The goal of the first technique is to mimic the representation from the

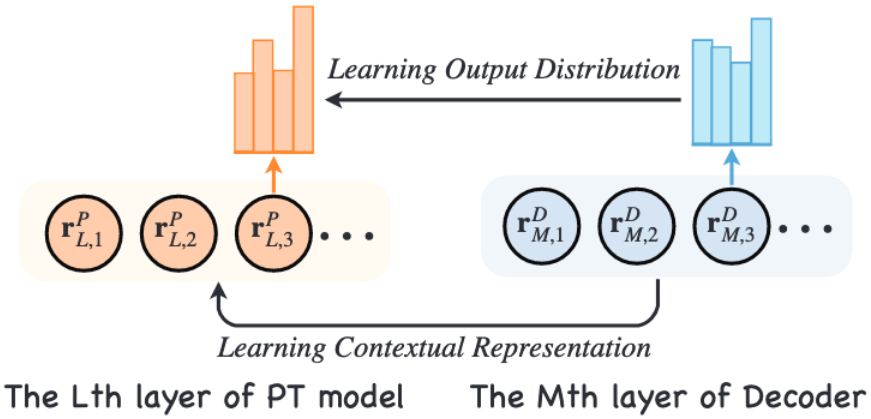

Figure 2.2: Illustration of knowledge distillation mechanism. Figure reprinted from [82].

pre-trained BERT by minimizing the cross-entropy loss. The second technique is introduced to perform a combination between the representation from BERT and the encoder of NMT. They achieve this by using a gating mechanism that uses the source as the input to decide how to fuse the representations. The intuition of this technique is that for a particular sentence, BERT might provide a better representation than the currently trained model. Using this gating mechanism, they try to dynamically adjust the use of BERT representation in the encoder. The intuition of this gating mechanism is that BERT might better encode some sentences better than others. The last technique uses a scheduling policy for adjusting the learning rate. They propose using this as they found that updating BERT LM and the NMT at different paces would benefit the final model. An illustration of the asymptotic distillation and dynamic switch mechanism is depicted in Figure 2.3.

[23] notice the discrepancy between the objective used to pre-train BERT and the objective used in common NLG tasks such as MT. They propose to introduce knowledge distillation approach learned in BERT for text generation tasks. The first proposal introduces a new objective called Conditional Masked Language Modeling (C-MLM). This objective is inspired by MLM but induces another conditional constraint while fine-tuning the pre-trained BERT on a target dataset. The

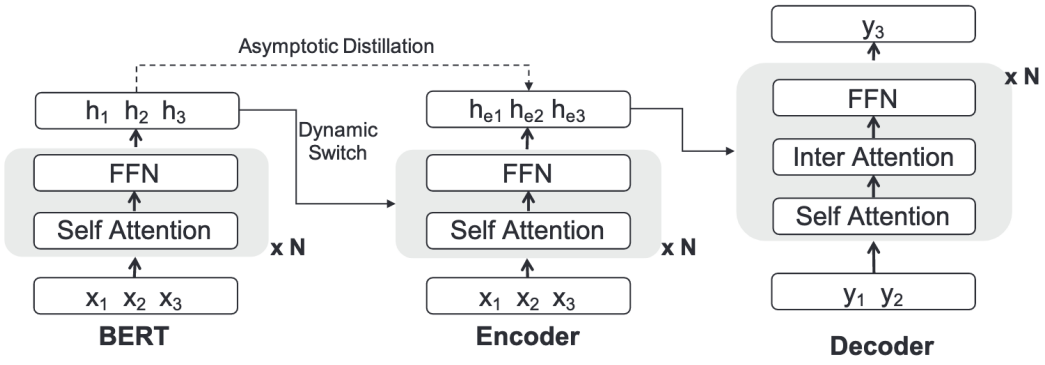

Figure 2.3: Illustration of asymptotic distillation and dynamic switch. Figure reprinted from [86].

second approach involves leveraging the fine-tuned BERT as a teacher model and using the logits output as the learning target for the student network to mimic. They claim that the proposed approach improves the generation output as they are now leveraging BERT's bidirectional ability to *plan ahead*. They also claim that BERT's ability to look forward into the future can act as an effective regularizer that can help to boost the quality of the generated output.

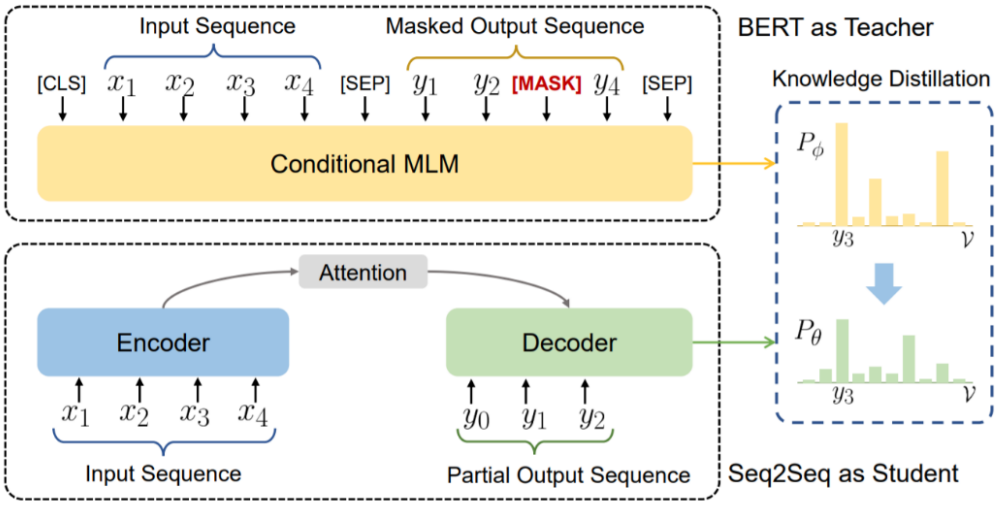

Figure 2.4: Illustration of seq2seq architecture. Figure reprinted from [23].

2.2 Adapters in Sequence-to-Sequence

In this section, we divide the discussions and reviews into three areas. Section 2.2.1 discusses the importance of adapters placement in transformer layers as well as in the seq2seq framework. The first part reviews the work of [36] and [3]. Both works propose the same concept of adapters with a similar architecture. The difference between them lies in the placement of the adapters within the transformer layer. The other work by [84] discusses the importance of adapters in the seq2seq framework. They found interesting results where adapters in the encoder have more impact than in the decoder.

Section 2.2.2 and Section 2.2.3 review the application of adapters in other fields such as NLU and ASR as well as in machine translation. Each will discuss the type of adapters used, the purpose of the adapters, and the impact of including the adapters as a part of their training procedure.

2.2.1 Placement of Adapters

There are two types of adapters placement in the transformer architecture. The first type of adapter was proposed by [36]. We recall that each transformer layer consisted of two sub-layers: a self-attention layer and a feedforward layer. Each sub-layers will be followed immediately by a projection layer whose job is to transform the feature's size back to the original input size. The adapters are applied to the output of the sub-layer of the transformer. Specifically, it is applied to the output of the projection layer that transforms the vectors back to the input size and prior to the skip connection. The illustration of the adapter architecture can be seen in Figure 2.5.

The second work of adapters is published by [3]. They take a simpler approach in contrast to [36] in regards to incorporating the adapters into the transformer network. Rather than inserting two serial adapters into the sub-layers of the transformer, they instantiate a single instance of adapters and place it on the top of

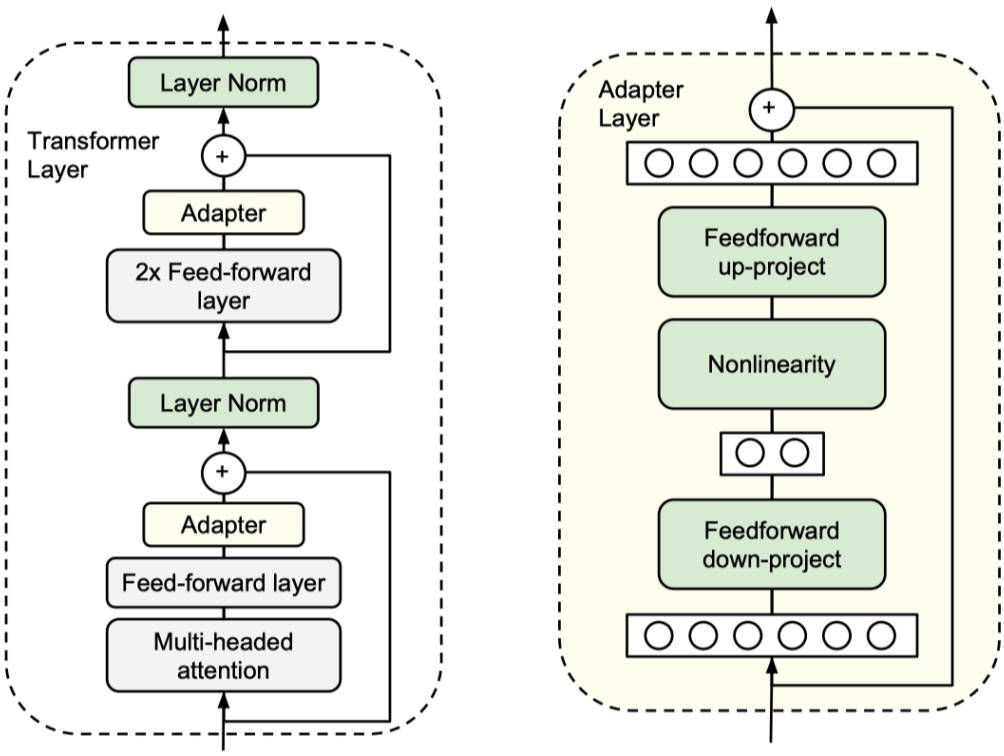

Figure 2.5: Illustration of adapter architecture. Figure reprinted from [36].

each layer. In addition to the simpler design of adapter layers, they add layer normalization to normalize the input of the adapters. The reasoning behind this is to make the adapters pluggable into any part of the base networks, ignoring the distribution variations of the previous layers. For illustration we refer to Figure 2.6.

[56] mentioned that the placement of adapters within a pre-trained model is non-trivial and thus requires extensive experiments. To identify the best setup for the placement of the adapter, the authors perform an exhaustive search on the hyperparameters. This includes the position and the number of adapters within a single layer, the position of residual connections, the bottleneck residual factors, as well as the non-linearity within the bottleneck adapter layer. In their experiments that involve both NLU and NLG, they find the best performing adapter is close to the simple architecture proposed by [3] and not from [36].

We now discuss the placement of the adapters from the perspective of the

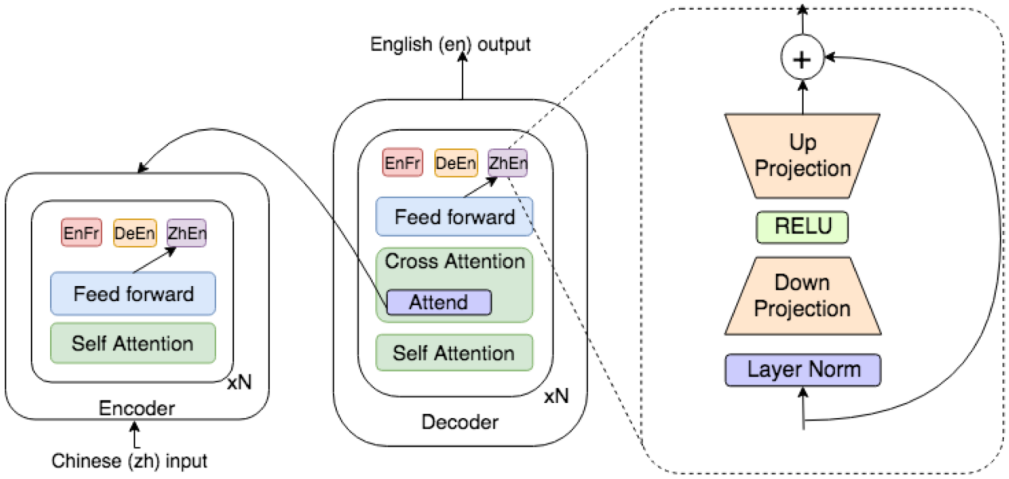

Figure 2.6: Illustration of adapter architecture. Figure reprinted from [3].

seq2seq framework. [84] conducted an experiment using similar adapter architecture and placement to [3] in ASR. The encoder in this work represents a model whose job is to encode audio representation. The decoder is then responsible for encoding the text features and generating text output. In this work, the authors perform experiments comparing the effectiveness of adapters in both the encoder and the decoder. They initialized the decoder using pre-trained mBERT weights [29]. Based on their findings, adapters in the decoder are not as effective as they are in the encoder component. These results indicate that adapting the weights in the audio space is more effective for the model's performance than in the text space. Features in mBERT may have already provided enough good features so that further fine-tuning may no longer be necessary.

2.2.2 Adapters in NLU and Automatic Speech Recognition

Adapters have been used as an alternative for naive fine-tuning in various fields in NLP. [36] introduces adapters to fine-tune BERT model in various NLU tasks. Primarily, they use the adapters in GLUE ([81]) and additional classification tasks. They also conducted another experiment on a more complicated problem such

as SQuAD ([64]). Compared to naive fine-tuning, they found positive results in using adapters in GLUE, other classification, and SQuAD tasks. By adding a small number of parameters, they achieved a comparable performance in all tasks compared to fine-tuning the whole weights. We can refer to Section 2.2.1 for the explanation and illustration of the adapters architecture of their work.

[58] experiments with adapters involve bootstrapping pre-trained NLP models such as BERT, and XLM ([27]) in low-resource languages. They perform two types of adaptation, language-adaptation and task-adaptation. For this purpose, they use two different adapters in different situations. The language-adapters are trained to capture various features in different languages with the MLM objective. They then put task adapters on top of the language adapters for further fine-tuning in each task. They use an efficient adapter architecture based on [56] which has similarities to the work of [3]. For more details on the architecture definition, we refer the reader to Section 2.2.1. They perform their experiments on three different tasks: Named Entity Recognition (NER), extractive question answering (QA), and causal commonsense reasoning (CCR). These tasks are available in multi-language test sets that cover both high-resource and low-resource languages. We refer the reader to their original paper for more details on the dataset they used on each task. Their main finding is that the task-specific adapters perform similarly to the work of [36]. However, since their main focus is on multilingual setup, they did not find satisfying results due to low output quality in unseen languages. With the help of language adapters, they have improved the performance across all tasks. The performance is especially appealing in low-resource languages.

[84] evaluate adapters performance in multilingual ASR setup. They use a similar setup as [3] to adapt both the encoder and decoder. They employ adapters to combat language mismatch and improve models' robustness in various languages with limited resources. Their main finding for the adapters is that it improves the performance across various languages. Furthermore, they also performed experiments to generate a cluster of languages where similar languages are put within

the same group. They then share the adapters to languages belonging to the same group and find out that this helps performance in low-resource languages but has small drawbacks in high-resource languages.

[46] propose multi-domain adapters for language model in ASR setup. Specifically, they employ the adapters in the language model (LM) component for rescoring purposes in an ASR system. They use a similar architecture of the adapter to the work of [36] where they introduce adapters within the sub-networks and at the top of the layer. Their experiments show that by using separate adapters for each domain, they can re-use a general domain LM and switch domains by replacing adapters with the one that has already been fine-tuned on the respective domain. They also found that with the help of adapters, they manage to improve the performance on a specific problem, such as predicting proper nouns.

2.2.3 Adapters in Machine Translation

[3] propose an alternative to [36]. Instead of using two different adapters within a transformer layer, they propose to use a single adapter on top of the layer. Another difference is that they add layer normalization after the transformer layer output. Furthermore, they also experiment with the adapters in the NLG domain instead of NLU. Instead of using BERT, they use machine translation objectives to pre-train the transformer model and treat the adapters as the adaptation module for a specific language pair. They first pre-trained the model in a large parallel corpus such as WMT before performing domain adaptation on a smaller corpus such as IWSLT in the same language pair. In a single language pair experiment, they found that the architecture of the proposed adapter is flexible with respect to the size of the data when the adapter's capacity is properly adjusted to match the requirements for the corpus size adaptation. Furthermore, they found that they did not observe any signs of overfitting; instead, they found the model to reach its peak and stay steady throughout the training process.

[59] apply adapters in multilingual machine translation differently than [3]. [3] use adapters for every target pair. For example, the English-French pair has to create two different adapters for English→French and French→English. [59] propose to use a monolingual adapter where instead of using a single adapter for a language pair, they use a single adapter for each of the languages. They use the same adapter architecture as in [3]. Their findings suggest that their approach reduced the number of adapters from $n(n-1)$ to $2n$, where n is the number of languages. Furthermore, they also find that this approach performs better in low-resource languages than the one proposed in [3].

[33] show that adding adapters to BERT during the fine-tuning can be beneficial in the machine translation task. The purpose of the adapters in this work is to make the fine-tuning process more lightweight than the regular fine-tuning. They follow the adapter architecture and use a similar mechanism to include the adapters on the base model as [3]. The difference is that [3] do not use BERT as the pre-trained model but instead use a plain transformer that is pre-trained using machine translation objective.

3. Experiment Setup

In this chapter, we describe our selection of datasets, framework, and automatic evaluation we used for the experiments. We start by describing our set of training and test data, our reasoning for choosing the dataset, and the tokenization algorithm in Section 3.1. We then move forward to the framework we use to implement the neural network, training, and evaluation phase in Section 3.2. Finally, we discuss the automatic evaluation we use during the experiments in Section 3.3.

3.1 German-to-English Dataset

The scope of our experiment is in a single language pair: German→English. We only select a single language pair because we want to focus our experiment on understanding the behaviour of BERT and the adapters in machine translation, not on generalization across multiple language pairs. We select IWSLT14 and WMT19 as our primary datasets. IWSLT14 will be mainly used as the dataset for fine-tuning and testing the final performance of the model, while WMT19 is used for the additional dataset in pre-training as well as in training some of our baselines.

3.1.1 IWSLT 2014

The 2014 IWSLT evaluation ([22]) is the fourth instance of a shared task started in 2010. This task is mainly focusing on the translation of TED Talks. The task is based on a collection of public speeches that cover various topics. All the talks in the collections have English captions. These captions are then translated into many languages by various volunteers worldwide. As TED talks are recorded events of speakers sharing their thoughts and experience, we need to deal with spoken language rather than written language. Spoken language is expected to be less complex and formal than written language.

Both the in-domain training and development data are available through the WIT3 website.[1] There is also out-of-domain training data provided through the workshop website. We focus only on the English German pairs in this work and ignore the other available languages. The evaluation dataset (tst2014) comprises talks from the previous years, and the 2014 talks are included in the training sets. Furthermore, to improve the reliability of assessing the MT progress over the years, evaluation sets from previous years (tst2013) are also distributed together with tst2014. On the other hand, development sets (dev2010, tst2010, tst2011, and tst2012) are kept intact from the past editions.

Evaluation sets tst2014 for the German language (De→En) are derived from the ASR task. Therefore, it is ensured that no overlap exists with other tasks that employ TED talks. In addition to the TED talks, there is a series of independent talks called TEDx. The difference lies in the location of the events. TED talks mainly focus on the North American region, while TEDx can be held in various areas worldwide. The TEDx-based corpus was proposed in 2013 for De→En as an additional test set to put more rigour into the evaluation process. Finally, a concatenation of the TEDx and TED-based development sets consisting of dev2010, tst2010, tst2011 and tst2012 sets were released. The complete statistics of the dataset can be seen in Table 3.1.

[1] `https://wit3.fbk.eu/`

	set	sentences	tokens	
			En	De
	train	172k	3.46M	3.24M
dev	TED.dev2010	887	20,1k	19,1k
	TED.tst2010	1,565	32,0k	30,3k
	TED.tst2011	1,433	26,9k	26,3k
	TED.tst2012	1,700	30,7k	29,2k
test	TED.tst2013	993	20,9k	19,7k
	TED.tst2014	1,305	24,8k	23,8k
	TEDx.dev2012	1,165	21,6k	20,8k
	TEDx.tst2013	1,363	23,3k	22,4k
	TEDx.tst2014	1,414	28,1k	27,6k

Table 3.1: Statistics of IWSLT 2014 German→English dataset.

3.1.2 WMT 2019

WMT19 dataset was first introduced in The Fourth Conference on Machine Translation (WMT) held at ACL 2019 ([5]). The primary objectives of this conference are to evaluate the state-of-the-art models in MT, advertise public access to the MT models' performance and the common test sets, and improve the methods to evaluate and estimate machine translation. There are various shared tasks within the conference that evaluate different machine translation aspects. This conference has been conducted 13 times and the current conference is built on top of the previous editions ([43, 16, 17, 20, 18, 21, 19, 7, 8, 12, 10, 9, 13]).

The dataset was collected from various news sources on the Internet. As mentioned before, we use WMT for the pre-training dataset and an additional dataset to train our baseline models. For this reason, we are not utilizing the dev and test set. Therefore, we show the statistics of the dataset in Table 3.2 only for the training set. Apart from a large number of sentences, another reason for choosing WMT19 as our additional dataset is that it contains sentence pairs from various domains.

corpus	sent	tokens	
		De	En
Europarl Parallel Corpus	1,8M	48,1M	50,5M
News Commentary Parallel Corpus	0,3M	8,3M	8,2M
Common Crawl Parallel Corpus	2,3M	54,5M	58,8M
ParaCrawl Parallel Corpus	31,3M	559,3M	598,3M
EU Press Release Parallel Corpus	1,4M	29,4M	30M
WikiTitles Parallel Corpus	1,3M	2,8M	3,2M

Table 3.2: Statistics of WMT 2019 German→English dataset.

3.1.3 Tokenization

BERT uses the subword tokenization algorithm called WordPiece ([70]) to construct the list of vocabularies. The algorithm is very similar to Byte-Pair Encoding (BPE; [72]), where it relies on a pre-tokenizer to split words within the training data, such as simple whitespace tokenization.

After the pre-tokenization, the set of words with their frequency is gathered. BPE then starts building a symbol vocabulary that consists of all symbols within the corpus. The symbol can be anything from alphabetical, numeric, and other symbols. BPE then learns a set of rules to merge and form a new symbol from two other symbols from the existing vocabulary. This process is repeated until the size of vocabulary items matches the desired vocabulary size.

To provide an example, let us assume that we have the following words and their frequencies[2]:

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

From these words, we have the set of initial symbols: "["b", "u", "n", "p", "h", "g", "s"]". BPE then starts the merging process by using the total frequency of each possible symbol pair. The pair that occurs the most often will be picked as a new vocabulary item. In our example, we have "h" followed by "u" with a total of 15 occurrences (10 times in `hug` and 5 times in `hugs`) and "u" followed by "g" with

[2]The example is taken from `https://huggingface.co/docs/transformers/tokenizer_summary`

a total of 20 occurrences. Therefore, we pick "ug" and append the new symbol to the vocabulary. We repeat this process until we meet the desired number of vocabulary items.

During the decoding process and assuming the following set of unique symbols: "["b", "u", "n", "p", "h", "g", "s", "ug"]", we now perform the tokenization process by matching the sub-word of the input word to the existing vocabulary. For example, if the incoming word is "mug", we will have ["[UNK]", "ug"] as our tokenization as we do not have "m" in our vocabulary. "[UNK]" is introduced as a special symbol to handle tokens/symbols that do not exist in the vocabulary. On the other hand, the word "bug" will be tokenized into ["b", "ug"].

3.2 Framework

3.2.1 HuggingFace Transformers

Transformers ([85]) is a library created by the Huggingface team that implements various transformer-based architectures. This library's primary aim is to implement transformer models specifically designed for research and production. This implies that the library is easy to read, extend, and supported by the industrial-strength implementation for deployment in production. The library also aims to facilitate transformer-based pre-trained model distribution to the public. Under the same foundation, this library supports the distribution and re-use of various pre-trained models in a centralized hub. This includes the configuration, such as the hyperparameters used to instantiate the models and the pre-trained weights. This improves research reproducibility as numerous users can now re-use and improve their experiments based on the pre-trained models.

The fact that the library is continuously maintained by the Huggingface team and supported by over 400 external contributors is one of our reasons for choosing Huggingface to conduct the experiments in this work. The library is released under

the Apache 2.0 license and is freely available on GitHub and their official website.[3] Furthermore, the website also provides easy-to-understand tutorials and detailed documentation of the API.

3.2.2 AdapterHub

Another reason we choose Huggingface is the availability of AdapterHub ([57]). Despite adapter's simplicity and achieving strong results in multi-task and cross-lingual transfer learning ([56, 58]), reusing and sharing adapters was not straightforward. Adapters are rarely released independently due to their subtle differences in architecture and strong dependence on the base model, task, and language. AdapterHub was created to facilitate the easiness of training models with adapters and share the fine-tuned adapters in various settings to mitigate these issues.

3.3 Automatic Evaluation

Bilingual Evaluation Understudy or `BLEU` ([52]) is one of the evaluation metrics that is widely used to evaluate MT models. It works by evaluating the output of an MT system (the hypothesis) with a set of manually translated references.

BLEU works by measuring the percentage of the matching n-grams in the hypothesis and taking the difference in length of the hypothesis and references as a form of penalty. The percentage of n-grams is often interpreted as a measurement of precision. However, in some cases, simply calculating the matching n-grams may lead to a misinterpretation of the output. In the case of unigram ($n = 1$), we can see the matching n-grams as the number of tokens available in the hypothesis and the references divided by the total number of tokens. See the following example:

Hypothesis: you you you you

Reference: I think you should know that you are right

[3] `https://github.com/huggingface/` and `https://huggingface.co`

The above example will lead to 100% unigram precision of the hypothesis despite the result only sharing the word **you**. To mitigate the precision problem, the shared number of n-grams in the hypothesis and the reference must be clipped by the number of n-grams in the reference. The following is the updated BLEU formula after incorporating the n-gram clipping:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n \in C} Count_{clip}(n)}{\sum_{C' \in \{Candidates\}} \sum_{n' \in C'} Count(n')} \tag{3.1}$$

Where n is the set of possible n-grams from a particular sentence used when calculating the score.

We mentioned that besides n-grams, BLEU also considers the length of the hypothesis as a penalty score. This is useful when dealing with short candidates within the hypothesis. The following is the example that shows the problematic output:

Hypothesis: you

Reference: I think you should know that you are right

Despite having a 100% precision score, the hypothesis does not represent the correct translation compared to the reference. We want to elude such a problem by introducing a penalty called **brevity penalty**. The penalty works by measuring the length of the hypothesis relative to the reference length and $e^{(1-r/c)}$ when the candidate is shorter than the reference length. To be more specific, we refer to the equation below.

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{3.2}$$

In the case of more than one reference, r is the length of the reference with the closest length to the hypothesis, called the **effective reference length**. One must note that the choice of reference will vary between different implementations

of BLEU. To be more precise, take the following example that shows the difference between two references is equal to 1, where one reference is one word shorter than the hypothesis and the other is one word longer:

Candidate: I eat fish on the beach with

Reference 1: I eat fish on the beach with friends

Reference 2: I eat fish on the beach

Combining the above components, BLEU is defined as follows:

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{3.3}$$

We define w_n as the weights for each n-grams scores (p_n) where $\sum_{i=1}^{\infty} w_i = 1$. w_n is usually set uniformly across all n depending on the number of n-grams used in the calculation. When 4-grams is used, it is set to $\frac{1}{4}$.

By default, BLEU computes the n-gram precision from unigrams to 4-grams. We perform the final score computation by calculating the average for each n-gram score and multiplying them by the brevity penalty (BP).

In this work, we follow the implementation of sacrebleu[4] ([60]) that is wrapped under Huggingface Transformers framework.

[4] `https://github.com/mjpost/sacreBLEU`

4. Adapters in Machine Translation

This chapter aims to study the impact of pre-trained models when fine-tuning machine translation models with adapters. Specifically, we are interested in understanding the contribution of good representation in the pre-trained language model to the adapters during fine-tuning. Furthermore, we are also interested in understanding the capability of adapters to perform with artificially degraded pre-trained models. We propose to use transformer-based architecture such as BERT (pre-trained or trained from scratch) for both the encoder and decoder components while using adapters for fine-tuning. To be more specific, we divide the experiments into four different pre-trained model setups:

- Use BERT weights[1] as the pre-trained weights (`Pre-trained BERT`). We use this model as the baseline for our experiments in this chapter.
- Use randomly initialised BERT models and pre-train the models with MLM objective on IWSLT and WMT data (`Pre-trained Transformer`). This setup is used to understand the impact of different data volumes and domains on the quality of the pre-trained models that will be fine-tuned with adapters in the later phase.

[1]We use publicly available BERT weights from Huggingface hub `https://huggingface.co`

- Use the pre-trained BERT model where the weights within the same layer are shuffled (`Pre-trained shuffled`). We use this setup to understand whether the adapters can capture important features from the original BERT weights and restore the performance even though the weights are no longer in the original position.

- Use randomly initialised BERT models with no pre-training (`Pre-trained random`). We use this setup as a complement of `Pre-trained shuffled` experiments to understand the impact of pre-trained models that contain relatively inferior knowledge than the original BERT model on the adapters when fine-tuned in the MT task.

4.1 Experiments Setup

4.1.1 Language Model

Dataset

From [29], we understand that BERT was trained with billions of words from various sources and domains. However, we do not fully understand the proper condition to stop adding more sentences to the pre-training data so that we can reduce the hours of training the model. Additionally, we do not know the impact of combining different domains in the pre-training data on the final performance of the model. For those reasons, we are reducing the scope of the pre-training data by restricting the creation of pre-training data only from machine translation datasets in two different domains and constructing the pre-trained models with different sizes of data. We use WMT and IWSLT for the experiment as WMT and IWSLT contain different domains, and WMT has a significantly larger volume and longer sentences. Specifically, we construct three different datasets with different volumes:

1. A standalone IWSLT.

Dataset	Initial Weights	Used for pre-training?	Used for fine-tuning?
Standalone IWSLT	Pre-trained BERT	No	Yes
	Random BERT	Yes	Yes
IWSLT + WMT 500k	Random BERT	Yes	No
IWSLT + WMT 2m	Random BERT	Yes	No

Table 4.1: The initial weights represent the weights used on both the encoder and decoder. Pre-trained BERT refers to the pre-trained models that use BERT. Random BERT models use BERT configuration but not the weights. Therefore, contrary to the Pre-trained BERT, Random BERT models undergo a pre-training process before fine-tuning. The pre-trained and fine-tuned columns represent the boolean value to mark whether the models initialized with the targeted weights are pre-trained or fine-tuned with the respective data. For example, Huggingface BERT weights are not pre-trained with IWSLT but use IWSLT for fine-tuning.

2. A combination of IWSLT and WMT data with a total of 500k sentences. We add the WMT data on top of IWSLT to increase the volume of our pre-training data. We should also note that the domain of WMT differs from the IWSLT, and we only test the models on the IWSLT test set. The WMT data can thus lead to worse translation performance due to domain mismatch.

3. A combination of IWSLT and WMT data with 2 million sentences. The same as (2) but with larger volumes.

Datasets (2) and (3) were constructed with a simple approach by using all training data from IWSLT, randomly selecting sentences from the WMT dataset, and combining them until we met the required number of sentences mentioned in the previous paragraph.

To illustrate how the datasets are used in the experiments, we refer to Table 4.1. We can see that datasets other than standalone IWSLT are used only for pre-training. The IWSLT is always used for fine-tuning, and depending on the initial weights, we also use it as pre-training.

Model

To pre-train the model, we follow the work of [29] by using the Masked Language Model (MLM) objective. In every sentence, some words will be masked, and the model has to predict the original words. A complete illustration can be found in Figure 4.1.

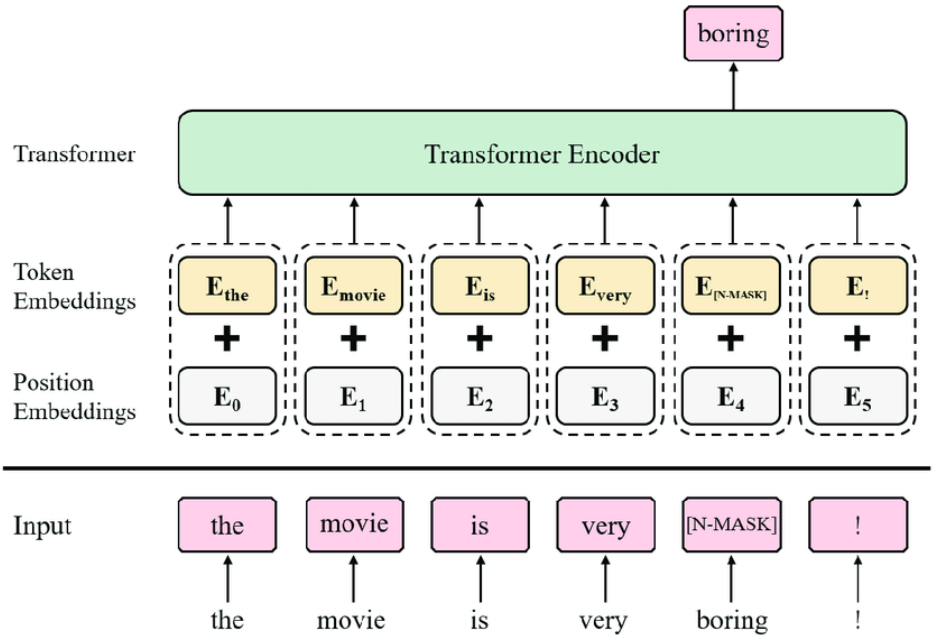

Figure 4.1: Illustration of the MLM objective during the pre-training. The illustration is reprinted from [53].

The MLM is introduced as an alternative objective to the auto-regressive language model. The auto-regressive objective is inherently different from the MLM. In auto-regressive, the models have to predict the word at step t, and they only have the ability to look at the previous words $(t-1...0)$ as the context. Auto-regressive is used in a couple of works such as GPT models ([15, 63, 62]) for pre-training. One of the advantages of using the MLM compared to the auto-regressive is that we can exploit the bidirectional context rather than only predicting the words from left to right.

We use the default BERT configuration from Huggingface to construct the model. The complete list of hyperparameters can be found in Table 4.2.

Name	En	De	Description
vocab_size	30522	31102	Vocabulary size of the BERT model
hidden_size	768	768	Dimensionality for each of the layers
num_hidden_layers	12	12	Number of hidden layers in the transformer
num_attention_heads	12	12	Number of attention heads for each attention layer
intermediate_size	3072	3072	Dimensionality of the feed-forward layer
hidden_act	gelu	gelu	The activation function within the layer
learning_rate	0.0005	0.0005	Learning rate used for the experiments
optimizer	adam	adam	Model optimizer
batch_size	64	64	Batch size used for the experiments
hidden_dropout_prob	0.1	0.1	The dropout probability for all fully connected layers in the embeddings, encoder, and pooler
attention_probs_dropout_prob	0.1	0.1	The dropout ratio for the attention probabilities

Table 4.2: Transformer BERT hyperparameters for English (based on `bert-base-uncased`) and German (based on `bert-base-german-dbmdz-uncased`).

We train the model until convergence with a maximum of 500 epochs. We define convergence by training for at least one day, with the validation loss no longer improving. We train the models in the MetaCentrum cluster with a specific configuration where the running jobs will be automatically killed after 24 hours. The loss usually starts to converge in less than 24 hours, but in some cases, we still see some models that are not yet converged. In this case, we continue the training by running another job process and loading the last checkpoint. On the other hand, if the loss starts to diverge before 24 hours or 500 epochs, we stop the training manually and select the checkpoint with the lowest loss score. Each language may end up converging in a different number of steps.

4.1.2 Machine Translation

Dataset

In machine translation experiments, there are several scenarios in which we use both WMT and IWSLT datasets. The IWSLT dataset is mainly used to fine-tune and evaluate the final model. The WMT, on the other hand, will be combined with IWSLT for training the baseline models. We use three different datasets for the experiments similar to the language model setup: IWSLT standalone, IWSLT + WMT (500k), IWSLT + WMT (2 million). To be more specific, we use the IWSLT standalone dataset for training, evaluation, and testing. The remaining data setups are used only for training, and we limit ourselves to the IWSLT data for evaluation and testing.

Model

We use the seq2seq architecture by [78]. The seq2seq architecture contains two different components, the encoder and the decoder. This architecture's details and illustration have been described in Section 1.2.3. The encoder and decoder use the same model and hyperparameters as described in Section 4.1.1. The only

modification made from the original language model is on the decoder side. On the encoder side, we only use the self-attention layers to gather some context from the neighbours within the same layer. On the other hand, the decoder requires further context to generate its outputs by including the vector representation from the encoder as additional features. For this reason, an extra component named `cross_attention` layer is introduced in the decoder, and it is trained from scratch in all experiments.

4.2 Experiments Results

This section discusses the results obtained for machine translation. First, in Section 4.2.1, we conduct the comparative study of using adapters in different pre-trained scenarios. Second, in Section 4.2.2, we perform a study by replacing the BERT model with a different BERT version where the weights are shuffled. We continue the experiments using pre-trained models with completely random weights and no pre-training. Finally, in Section 4.2.3, we perform experiments comparing the performance of the randomly set weights pre-trained model that is fine-tuned with adapters with the baseline models as well as models that are pre-trained with the combination of WMT and IWSLT.

4.2.1 Adapters Comparison

Experiment Setup and Motivation

In this section, we conduct experiments to understand the contribution of adapters by comparing trained and fine-tuned models with different sizes of datasets. The definition of the dataset is the same as have already explained in the previous section. The dataset is used for two different purposes:

- Used to train the BERT-style language model from scratch, separated for source and target languages.

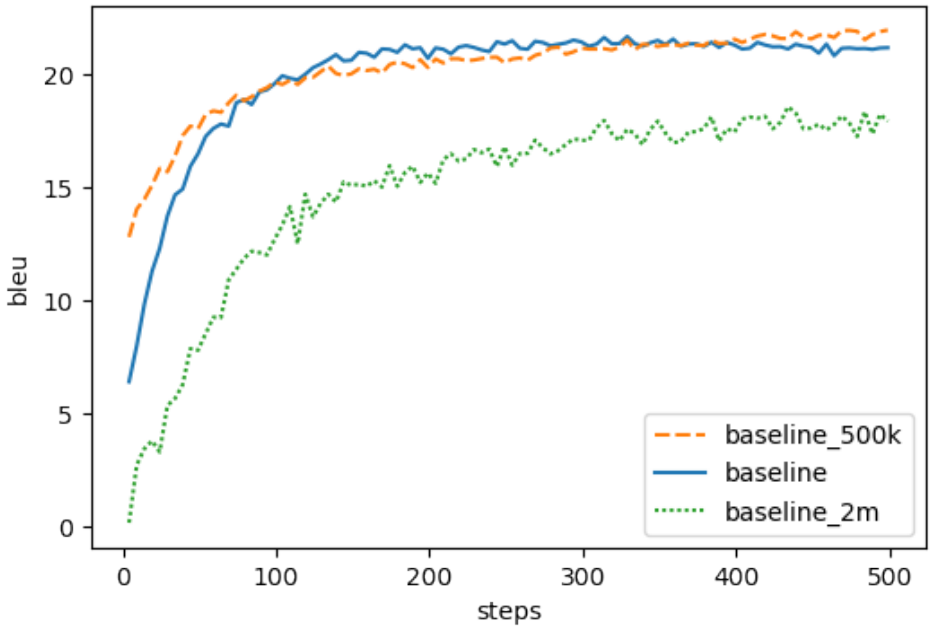

Figure 4.2: Comparison between baseline models trained with different size of datasets. `baseline` represents the model trained only using IWSLT; `baseline_500k` represents the model trained using IWSLT and WMT with total of 500k sentence pairs; `baseline_2m` represents the model trained using IWSLT and WMT with total of 2 million sentence pairs.

- Used for pre-training and later fine-tuning the seq2seq model on IWSLT parallel data.

Experiment Results

In this section, we compare the result of the baseline models with the fine-tuned models with adapters. Our first experiment compared the results from an empty BERT model (a transformer model with BERT configuration but no pre-trained BERT weights) without any pre-training and adapters and only trained the model from scratch. We refer to these models as our `baseline` for the rest of this chapter. From Figure 4.2, we can see that adding more data to the baseline models does not necessarily improve the performance. We suspect the models require more training time to get the best final performance. There is a clear gap between `baseline_2m` and the rest of the baseline models. `baseline` and `baseline_500k`

perform really well from the start while `baseline_2m` lags behind. We suspect this is due to the effect of including more sentences from different domains. We found that `baseline_500k` performs the best as it balances the models to avoid overfitting in the intended domain and still benefits from out-of-domain data. `baseline_2m` shows the impact on domain difference where it has relatively lower performance than the other models. However, we must acknowledge that the model has not fully converged in the limited training time (measured by the number of steps) we have and has a chance to improve its performance further. Despite the lower performance in BLEU score, `baseline_2m` deserves a manual evaluation of the translation output. We argue that in some cases, lower BLEU may not necessarily reflect a lower quality. We perform the manual evaluation at a later stage in this chapter.

To see the impact of adapters, we compare the result on different sizes of pre-training data used for the base model. The base models are then fine-tuned with the adapters module on the IWSLT data. As we can see from Figure 4.3, pre-trained BERT with adapters achieves the best performance from the earlier steps compared to the rest of the models. In contrast to the baseline models, we see the benefit of adding more sentences to the pre-training (see `adapters_pt_2m` vs `adapters_pt_500k`). From Figure 4.2 we see that `baseline_500k` outperforms `baseline_2m` by approximately 5 BLEU despite larger training data on `baseline_2m`. Note that the training data we refer to in the baseline models is the training data for MT. On the other hand, when we add more training data on the pre-training side, we can see that the model trained using 500k data has a lower performance than the one using 2 million data. This tells us that adding larger pre-training monolingual data and then fine-tuning the models with adapters impact the performance positively.

In the model that only uses IWSLT as the pre-training data, we observe performance degradation in the middle of the fine-tuning. We found this is due to the gradient explosion in the cross-attention layer, and we attribute this instability

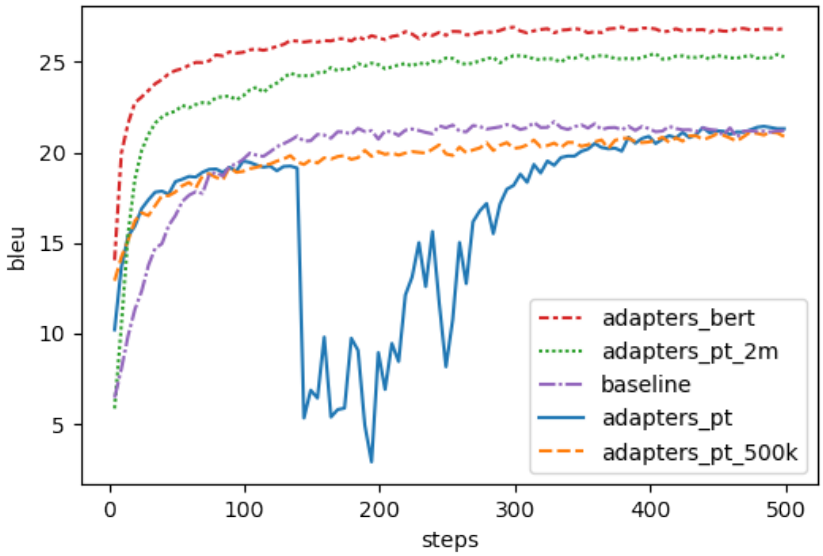

Figure 4.3: The effect of pre-training data size when the adapters are then trained on the same IWSLT data. `baseline` represents the model trained only using IWSLT; `adapters_pt` represents the model pre-trained only using IWSLT; `adapters_pt_500k` represents the model pre-trained using IWSLT and WMT with total of 500k sentence pairs; `adapters_pt_2m` represents the model pre-trained using IWSLT and WMT with total of 2 million sentence pairs; `adapters_bert` represents the model that uses BERT weights.

to the small data used in the pre-training. In the later steps, the IWSLT model eventually achieved performance similar to the 500k model.

4.2.2 Random and Shuffled Pre-trained Weights

Experiment Setup and Motivation

We perform two different categories of experiments to study how the adapters benefit from the pre-trained weights. First, we conduct experiments where we shuffle the BERT weights. We separate the weights initialization into two approaches to perform the experiments:

- We shuffled the weights from the column perspective. We shuffled the column-wise order of all the matrices while keeping the row-wise order intact.
- We shuffled the weights from both column and row perspectives.

Second, we conduct experiments where we set random weights on all base network layers and treat them as the pre-trained model. During the fine-tuning, we only update the adapter weights and keep the random weights intact.

Experiment Results

We can see from Figure 4.4 that the performance of the model that uses random pre-trained weights is more stable than the one using shuffled BERT weights. All shuffled BERT models suffer from gradient explosion similar to the IWSLT model we show in the previous section. Furthermore, shuffling the weights on both the row and column sides seems more detrimental than just shuffling on the column side. This may show that there could be some pattern in `adapters_shuffled` that the adapters can eventually help to recover during the fine-tuning, while on `adapters_shuffled_both` more patterns are missing and more challenging to recover.

Although the performance of the random model (`adapters_random`) is still below the baseline model, it is interesting to see that fine-tuning only the adapters, cross-attention, and output layers is sufficient to achieve a reasonable BLEU score, considering that the pre-trained model does not contain meaningful information relative to the original BERT weights. Furthermore, it is also interesting to see that compared to `adapters_shuffled_both`, we can get better final performance even though we can categorize `adapters_shuffled_both` as a model that uses randomly set weights pre-trained model. We hypothesize that this is probably related to the difference in the distribution of the weights between the models. In `adapters_random` we are limited to the distribution of the weights in the initialization algorithm used in BERT, while in `adapters_shuffled_*` the distribution may be different as the weights in pre-trained BERT have already been optimized with the MLM objective.

Since we are relying on the pre-trained model with a random set of weights and with no further training, there is a possibility that our method may only work

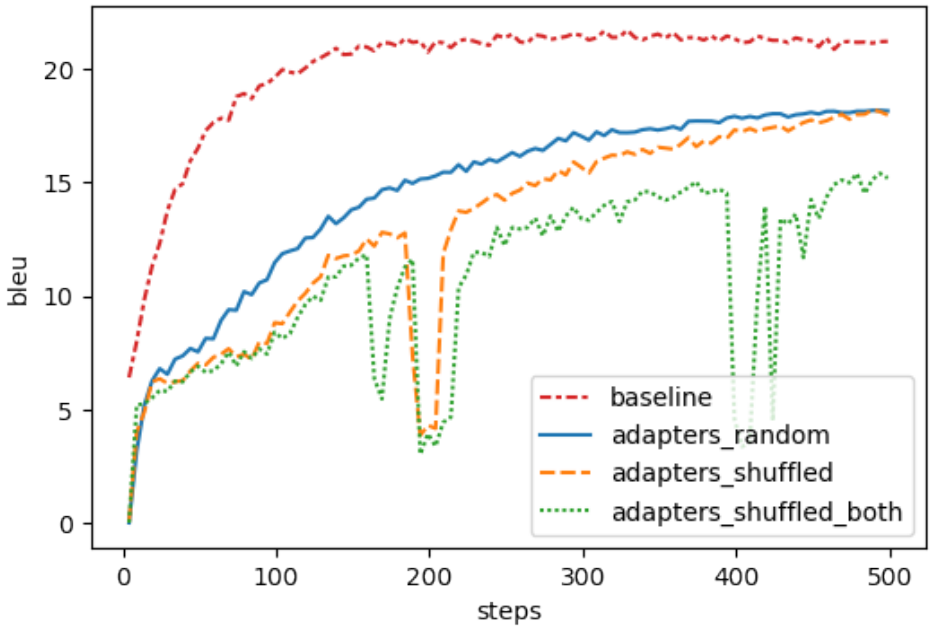

Figure 4.4: Comparison between adapters using shuffled BERT and random weights as the pre-trained models. `baseline` represents the model trained only using IWSLT; `adapters_random` represents the model pre-trained only using random weights; `adapters_shuffled` represents the model pre-trained using column-wise shuffled BERT model; `adapters_shuffled_both` represents the model pre-trained using shuffled BERT model.

in a single random seed. To ensure a robust experiment, we repeat the random experiments ten times with ten different random seeds. We can see from the result in Figure 4.5 that all the random seed performs similarly to one another.

4.2.3 Random Pretrained vs. Out-of-Domain Data

Experiment Setup and Motivation

In this experiment, we use the same setup as in Sections 4.2.1 and 4.2.2. Specifically, we are interested in investigating the model's performance that uses randomly set weights as the pre-trained model compared to the baseline model. We recall that we can gain a reasonable score when fine-tuning the model with adapters from the previous experiments using randomly set pre-trained weights on the transformer model. In this experiment, we are conducting further study to understand the

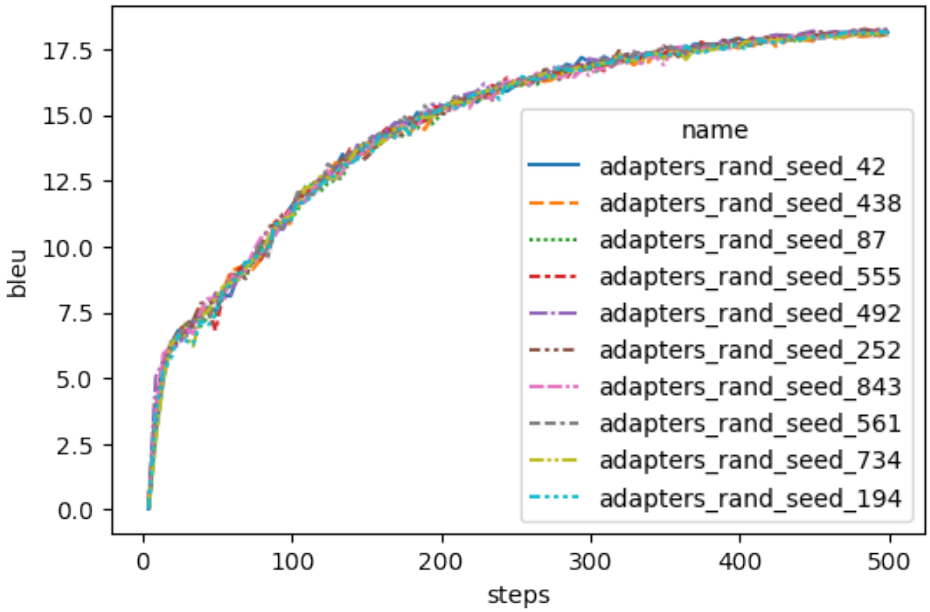

Figure 4.5: Comparison of different random seed for randomly set weights based models.

performance relative to the `baseline` model, where the model was trained using only IWSLT data and a different baseline (`baseline_2m`), where the model was trained using a mix of WMT and IWSLT. This comparative study aims to understand whether we can see the benefits of using adapters versus training the whole transformer model with bigger data sizes.

Experiment Results

We analyze the random weights by comparing the result with the best-performing baseline and our pre-trained transformer models. From Figure 4.6, we can see that the performance of the random model achieves a similar result as the baseline model that uses 2 million training sentence pairs. Recall that the baseline model is the BERT-style transformer model trained from scratch without fine-tuning and adapters. This tells us that training the whole model with bigger data does not necessarily improve the model's performance. It may need further tuning to gain the benefits of bigger data and bigger models because any departure from the

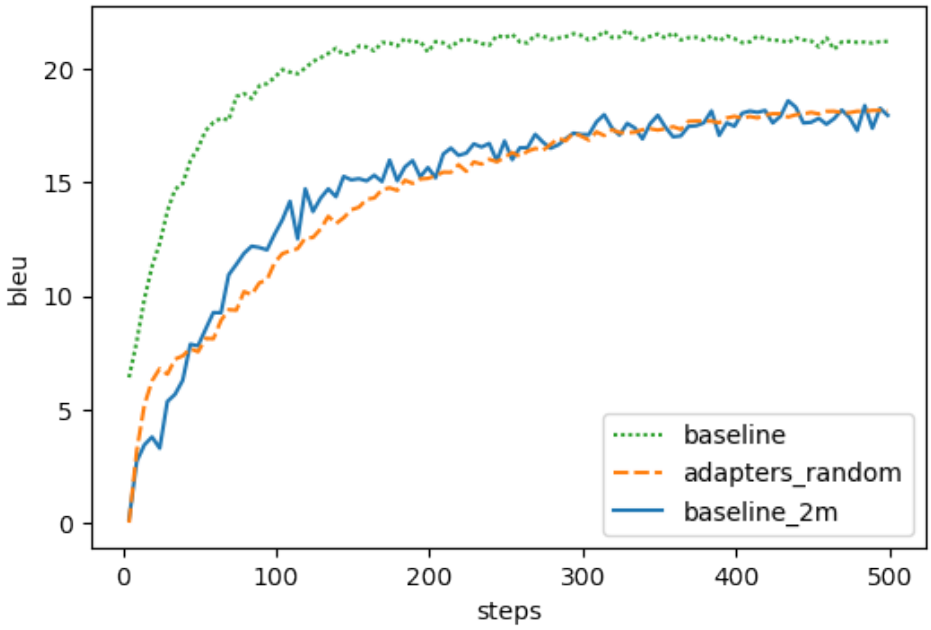

Figure 4.6: Comparison between pre-trained random weights and baseline model. `baseline` represents the model trained only using IWSLT; `baseline_2m` represents the baseline model trained with a combination of IWSLT and WMT sentence pairs; `adapters_random` represents the model pre-trained only using random weights.

domain of the test set can be more harmful than useful. We can see the result of using random weights as a potential alternative for training the model with small data such as IWSLT. While the performance is still far from the baseline (no adapters, full model training on IWSLT, i.e. in-domain data), this result shows that the base model's structure helps the adapter achieve a meaningful performance with a tiny portion of weights trained in the fine-tuning.

We perform a quick check of the output of the model in Table 4.3. From the first two lines, we can see that the model has difficulty capturing complex phrases. In the first row, the model missed **tanzen im tempel** which means **dancing in the temple**. For the second row, the model confuses **knowledge of the forest** and outputs **audience** instead. Furthermore, the model also does not translate the word **kenntnis** and makes the translation unclear since the object of the sentence is missing. Despite those mistakes, the model still can capture simple sentences, as shown in the third row. Another observation that we noticed in the generated

Random Weights + Adapters
Input: wir tanzen im tempel und werden zu gott. & quot ; **Reference**: we dance in the temple and become god . " **Hypothesis**: we & apos ; re going to be able to become god. & quot ;
Input: aber gleichzeitig hatten sie eine klare kenntnis des waldes, die erstaunlich war. **Reference**: but at the same time they had a perspicacious knowledge of the forest that was astonishing. **Hypothesis**: but at the same time, they had a clear of the audience who was amazing.
Input: es ist so wunderbar. ihr musst es beschutzen. & quot ; **Reference**: it is that beautiful . it is yours to protect . " **Hypothesis**: it & apos ; s wonderful. you have to protect it. & quot ;

Table 4.3: Prediction results from randomly set pre-trained model fine-tuned with adapters.

output is the tokenization of `&quot` and `&amp` . Instead of being treated as a single token, the tokenizer treats the token as three different subword tokens. We notice that this is due to the unavailability of the aforementioned token in the pre-trained BERT vocabulary.

4.3 Qualitative Comparison

We perform a manual check on the generated translations of some of our models. From Table 4.4, we can see that none of the models generated the correct result for the first example. However, BERT + adapters and 2 million pre-trained base models generate the proper context where the result is still discussing **the patient**. The wrong part is when the model generates an incorrect translation regarding the patient's disease. The second example shows that the BERT + adapters model creates a better quality of translation than the other models. The final example shows that BERT + adapters generate an interesting output where it manages to remove unimportant characters such as "`--`" and produce readable output. People may vary in their opinion on this example as the 2 million pre-trained base model

Sample Translation Output
Input: erinnerst du dich an den patienten mit dem gereizten rachen? **Reference**: do you remember that patient you saw with the sore throat? **Hypothesis 1**: do you remember reading to the patients? on the **Hypothesis 2**: do you remember the patient with the tingling revenge? **Hypothesis 3**: remember the patient with the bruised remorse?
Input: großartig, sagte ich. legte auf. **Reference**: great, i said. got off the phone. **Hypothesis 1**: great, i said. got up.. **Hypothesis 2**: great, i said. put on.. **Hypothesis 3**: great, i said. put it down.
Input: - - aber in unserer entdeckung der welt, haben wir alle arten unterschiedlicher methoden. **Reference**: but in our discovery around the world, we have all kinds of other methods. **Hypothesis 1**: but in our discovery of the world, we & apos ; ve got all sorts of different **Hypothesis 2**: - - but in our discovery of the world, we have all kinds of different methods **Hypothesis 3**: but in our discovery of the world, we have all sorts of different ways of doing things.

Table 4.4: Prediction results from **Hypothesis 1**: Baseline model trained with only IWSLT data; **Hypothesis 2**: Pre-trained model with adapters where we pre-train the model with IWSLT and WMT with a total of 2 million pre-training data; **Hypothesis 3**: BERT with adapters.

Input	Output
& quot ; moneyball & quot ; erscheint bald und dreht sich um statistiken und um diese zu nutzen ein großartiges baseball team aufzustellen.	& quot ; devilball & quot ; appears soon, and it & apos ;
moneyball erscheint bald und dreht sich um statistiken und um diese zu nutzen ein großartiges baseball team aufzustellen.	fatball soon appears and it turns out statistics and to use that to build a great baseball team

Table 4.5: An example of bad model behaviour when the input contains unknown tokens like `&quot` (top row). The translation is not abruptly cut if these symbols are removed (bottom row).

generates a more concise output.

We also notice that the BERT-based models have difficulties generating long sentences. The models always cut the translation short when an unavailable token such as `&quot` appears in the sentence. To give a more explicit example, Table 4.5 illustrates the significant difference in the outputs when the token `&quot` is observed in the input. This shows that tokenization plays an essential role in the model where tokens unavailable in the vocabulary list may affect the generated output significantly.

5. Adapters Effectiveness in Machine Translation

We continue the study from the previous chapter to understand more about the relation between adapters and pre-trained models. Similar to the previous chapter, we use BERT and its variants as the pre-trained models and fine-tune them with adapters. This study aims to evaluate the combination of adapters and BERT in machine translation and study the effectiveness of adapters by putting them only in the encoder or the decoder. We also experiment with down-scaling the pre-trained model size and try to recover the performance of the full-sized model. We separate the experiments into three different areas:

- Use BERT weights[1] as the pre-trained weights and investigate the importance of adapters in encoder or decoder.
- Use BERT weights and investigate their importance compared to random weights in the encoder or decoder while fine-tuning with adapters.
- Down-scaling BERT weights by either zeroing out half of BERT's weights (`zbert`) or completely removing them from the weight matrices, squashing the matrices (`zsbert`). We use the down-scaling technique to understand

[1]We use publicly available BERT model from Huggingface hub `https://huggingface.co`

whether we can use adapters to recover the performance of the original BERT (without adapters) while using fewer parameters.

5.1 Fixed Variable Parameters of Experients

5.1.1 Framework

As we mentioned at the beginning of the chapter, we have several scenarios we use to conduct the experiments. We start by describing the variables that we fixed throughout the experiments. As we have mentioned in Chapter 3, we use Huggingface as our main framework with added modifications for adapters. Contrary to Chapter 4, we do not investigate language models that we train ourselves, but instead, we focus only on the BERT language model.

The model and hyperparameters that we use throughout the experiment remain the same as described in Chapter 4. We use transformer model with seq2seq architecture and the BERT-based hyperparameter configuration to initialize both the encoder and the decoder.

5.1.2 Dataset

As mentioned in the previous section, our focus in this chapter is on machine translation. We use IWSLT dataset to perform the fine-tuning as well as the evaluation for the models.

5.2 Original BERT

5.2.1 Size of Adapters

Experiment Setup and Motivation

In these experiments, we freeze both the encoder and decoder and modify the reduction ratio parameter in the adapters. The adapter serves as a bottleneck layer

with two dense layers and a non-linear function between them. The reduction ratio is defined as the fraction of the original representation dimension divided by the adapter vector size. For instance, if we use 16 as the reduction ratio, we reduce the original layers by 16 with the first dense layer and then scale it back to the original size with the second dense layer.

We try out various sizes of reduction ratios to compare the results. This reduction aims to see whether we can further benefit from enlarging the adapters' bottleneck size. We use 16, 8, 4, 2, and 1 as the ratio values for this experiment. We compare the results with the baseline BERT that we fine-tuned by only training the cross-attention and output layers. We will refer to this baseline as `baseline_bert` for the entirety of this chapter.

Experiment Results

In this section, we compare the results of the `baseline_bert` with BERT models that are fine-tuned with adapters in different reduction ratios. We are fine-tuning the cross-attention and output layers for the `baseline_bert` model and freeze the rest of the model. We can see in Figure 5.1 that even the smallest model (`adapt_bert_reduc_16`) can already outperform the baseline by around 2 BLEU points. This shows that the adapters can help improve the model's performance by adding only a small number of weights during the fine-tuning.

Despite better performance than the baseline model, the difference between the ratios is minimal. It suggests that there is not much benefit in expanding the size of adapters for the normal size BERT. It is possible that it is no longer trivial to simply append larger-size adapters for fine-tuning the model and getting better performance. Further changes may be required to handle the different nature of BERT's output as it is naturally different from the common auto-regressive machine translation objective. Furthermore, we recall that we also have problems where the models struggle in generating good translations when the input contains an unavailable token such as `&quot` .

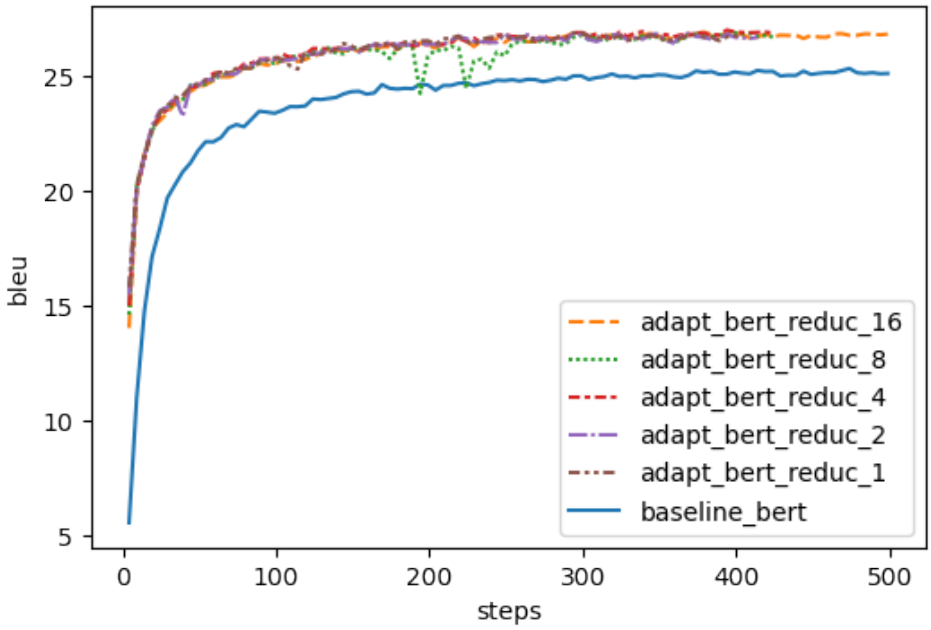

Figure 5.1: Comparison between baseline BERT model and adapters model with different ratio (16, 8, 4, 2, 1).

5.2.2 Position of Adapters (Encoder vs Decoder)

Experiment Setup and Motivation

We would like to see the importance of adapters when they are put in different places. Since we are working with seq2seq architecture in this work, we would like to see whether only incorporating adapters on either the encoder or the decoder can already be beneficial and reduce the number of parameters added to the model.

Experiment Result

We see in Figure 5.2 that adding adapters just in the encoder brings an improvement and outperforms the baseline. Adapters only in the encoder train the fastest at the beginning, and their final performance is almost the same as if we added the adapters on both sides. For the decoder, on the other hand, we can see that aside from a more promising start, there is no benefit as there is no improvement in terms of late BLEU scores compared to the baseline. With this finding, we can

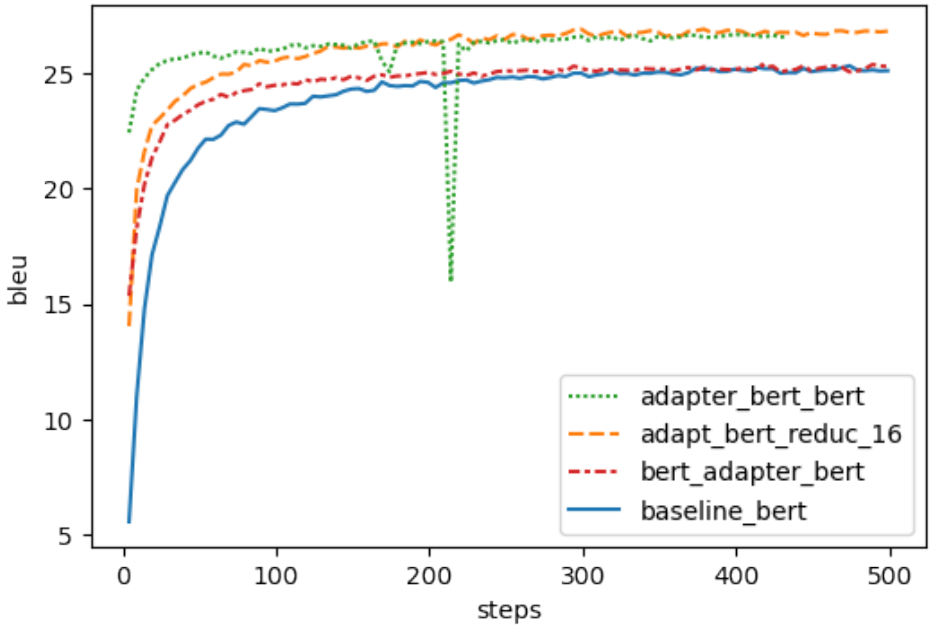

Figure 5.2: Comparison between baseline BERT model and adapters model where the adapters are placed in three different setups: 1) Adapters in both encoder and decoder (`adapt_bert_reduc_16`); 2) Adapters only in encoder (`adapter_bert_bert`); 3) Adapters only in decoder (`bert_adapter_bert`).

reduce the cost of fine-tuning further by half when we do not include the adapters on the decoder.

5.2.3 True BERT in Encoder vs Decoder

Experiment Setup and Motivation

This section investigates the importance of pre-trained BERT in the encoder or the decoder when the adapters are used for fine-tuning the models.

The previous chapter introduced an experiment where we instantiate the base transformer model with only random fixed weights. We are then fine-tuning the base transformer model by only updating the cross-attention, adapters, and output layers. In this setup, we are doing experiments in a similar concept. We use random (fixed) weights instead of the original pre-trained BERT weights in the encoder or the decoder. We thus have a seq2seq model with the random weights encoder followed by the BERT decoder and vice versa. In the fine-tuning stage, we update the adapters in the encoder and decoder, the cross-attention (or cross-attention layer only in our baseline models), and the output layers.

The purposes of the experiments are:

- We want to understand further the importance of the pre-training model when fine-tuning with adapters. By initializing the models with BERT only in one component, we can see whether it is necessary to use BERT on both components when adapters are incorporated.

- We want to understand the capability of adapters when either one of the components does not contain useful information (relative to BERT). We would like to see whether the adapters can recover or even outperform some of the performance that we have already gathered from the previous chapters and sections.

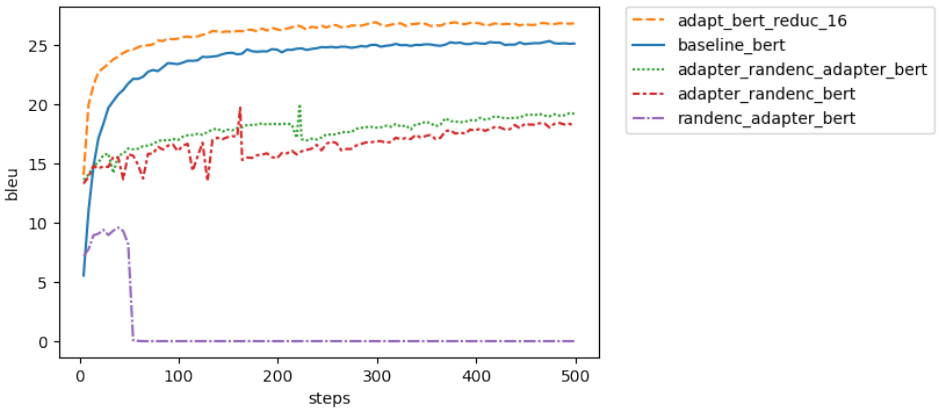

Figure 5.3: Random + BERT: Comparison between baseline BERT model and adapters model where the adapters are placed in three different setups: 1) Adapters in both encoder and decoder (`adapt_bert_reduc_16`); 2) Adapters only in encoder (`adapter_bert_bert`); 3) Adapters only in decoder (`bert_adapter_bert`) and the decoder is initialized with BERT while the encoder is initialized with random numbers.

Experiment Results

This section compares models that use adapters in either or both the encoder and decoder while only initializing one of these components with pre-trained BERT and the other one with (fixed) random weights.

Randomly Set Weights on Encoder In this part of the section, we want to answer the main question: "To what extent can the adapters restore the missing gap when the encoder does not contain useful information (relative to BERT)?"

We can see from Figure 5.3 that when adapters are used in both components to a model with random encoder weights (`adapter_randenc_adapter_bert`), we get to almost 20 BLEU points. This is relatively higher than the other two setups: adapters only in the encoder (`adapter_randenc_bert`) and only in the decoder (`randenc_adapter_bert`). However, compared to the baseline, we are missing 4 BLEU points when we set the encoder with completely random weights. This means that the base encoder model did contain relatively essential information that the adapters can not simply restore during the fine-tuning.

When the adapters are removed from the decoder (`adapter_randenc_bert`), we see a degradation in performance about 1 BLEU point compared to the model that uses adapters on both side (`adapter_randenc_adapter_bert`). However, when the adapters are removed from the encoder (`randenc_adapter_bert`), the performance is completely depleted to zero during the training. We also see the same behaviour in the next section when the weights on the decoder are set randomly. This tells us that it is not trivial to simply fine-tune the cross-attention without further modifying the encoder's parameters when the parameters on the encoder are completely random.

Randomly Set Weights on Decoder Similar to the previous section, the main question in this experiment is, "To what extent can the adapters restore the performance when the decoder does not contain useful information (relative to BERT)?"

In contrast to when the randomly set weights are on the encoder side, we can see from Figure 5.4 that fixing a random decoder leads to performance comparable to the one we have on `bert_baseline`. This tells us that the pre-trained weights in the encoder are more important than in the decoder when we have adapters on both sides. However, when removing the adapters on the encoder, we see similar performance as in the previous section, where the performance drops to zero in the middle of training. This further strengthens our argument that adapters are necessary to adjust the weights in the model so that the cross-attention layer can work properly.

On the other hand, when we remove the adapters from the decoder side, we can see that the performance is not as bad as when the adapters are removed from the encoder, but we still see a reduction in performance. We see a reduction around less than 1 BLEU point when the model reaches 400k steps in the training stage. It is possible that even with random weights on the decoder side, the adapters help the cross-attention layers produce a good vector representation with meaningful features for the decoder to generate reasonable translation outputs.

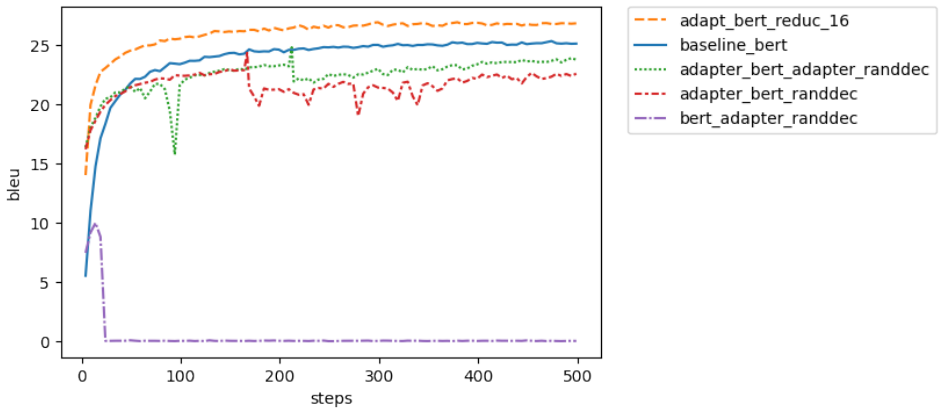

Figure 5.4: BERT + Random: Comparison between baseline BERT model and adapters model where the adapters are placed in three different setups: 1) Adapters in both encoder and decoder (`adapt_bert_reduc_16`); 2) Adapters only in encoder (`adapter_bert_bert`); 3) Adapters only in decoder (`bert_adapter_bert`) and the encoder is initialized with BERT while the decoder is initialized with random numbers.

5.3 BERT Size Reduction

5.3.1 Zeroing Columns

Experiment Setup and Motivation

In this experiment, we will focus on the soft reduction of BERT weights by zeroing the weight matrices on every even column and row indices within the transformer body as well as in the embedding. We load the pre-trained BERT weights, manually edit them and then continue the experiments by fine-tuning the cross-attention, adapters, and output layers. We refer to this setup as `zbert` for the rest of this chapter.

Besides removing the columns, we also perform experiments where we put the adapters either on the encoder or the decoder. This particular experiment aims to understand the model's behaviour when the pre-trained BERT is replaced with this particular setup.

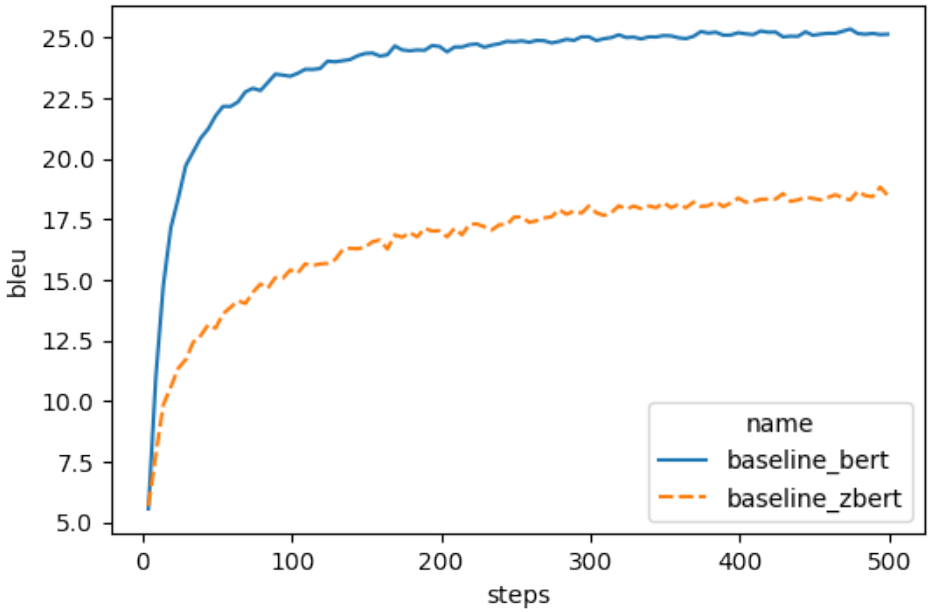

Figure 5.5: Comparison between baseline BERT model and baseline `zbert` models.

Comparison with BERT Baseline (Full BERT Fine-tuning)

We first compare the `zbert` model without adapters and only fine-tune the cross-attention and output layers. We use `zbert` weights on both the encoder and decoder so that it is comparable to the model that uses full-weight BERT. We use the full-weight BERT as the baseline in this experiment.

We can see in Figure 5.5 that we are losing performance of about 4 BLEU points. This is significant as we lose essential features from the original BERT model. To see whether we can recover some of the performance with adapters, we continue our experiment by fine-tuning the `zbert` model that is instantiated on both encoder and decoder sides with adapters. We can see from Figure 5.6 that we only managed to recover 1 BLEU point with a reduction ratio of 16.

From Figure 5.7, when we increase the size of the reduction ratio, initially, we can see some improvement in the BLEU score compared to the higher ratio. However, they eventually converge to a similar performance by the end of training with no significant difference between different ratios. From this result, we can understand that depending on the base pre-trained model, adapters still have a

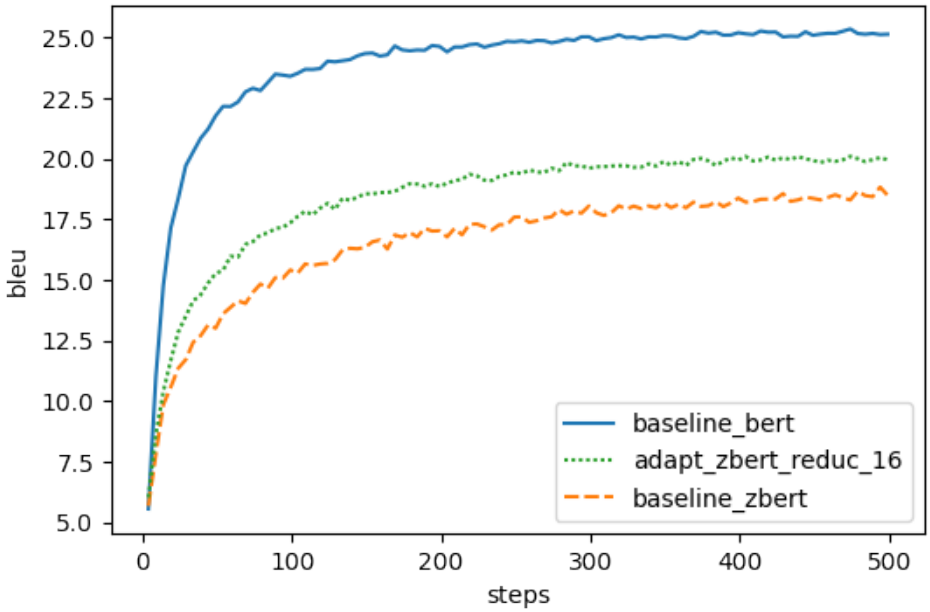

Figure 5.6: Comparison between baseline BERT model, baseline `zbert` and adapters `zbert` models.

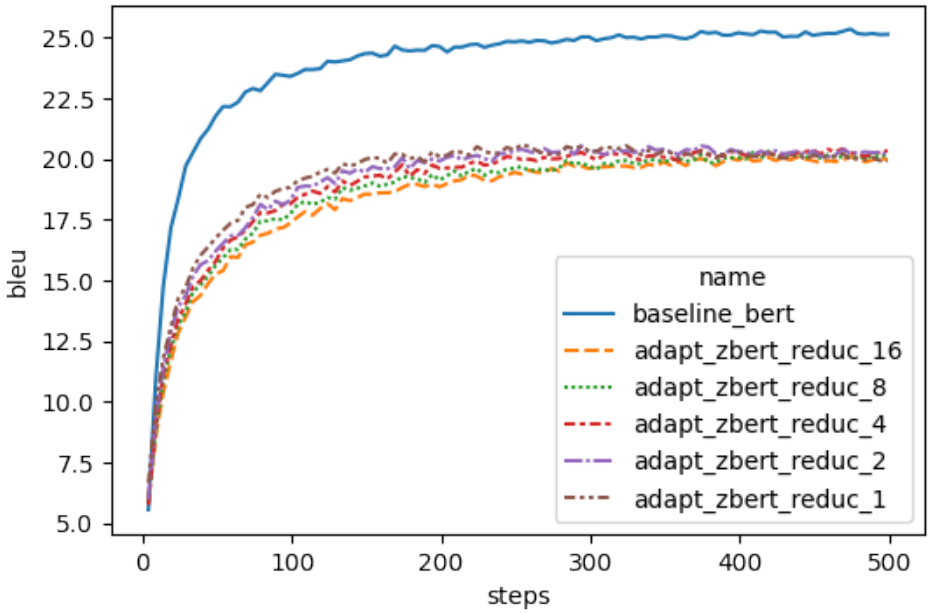

Figure 5.7: Comparison between baseline BERT model and different reduction ratio of `zbert` models.

limitation in achieving certain performance.

Adapters Position In this section, we aim to understand whether the position of both adapters and the pre-trained models affect the model's performance, similar to what we have seen in Section 5.2.2. We use a similar setup as in the previous section, with the exception that we use `zbert` as the pre-trained model instead of the original BERT model.

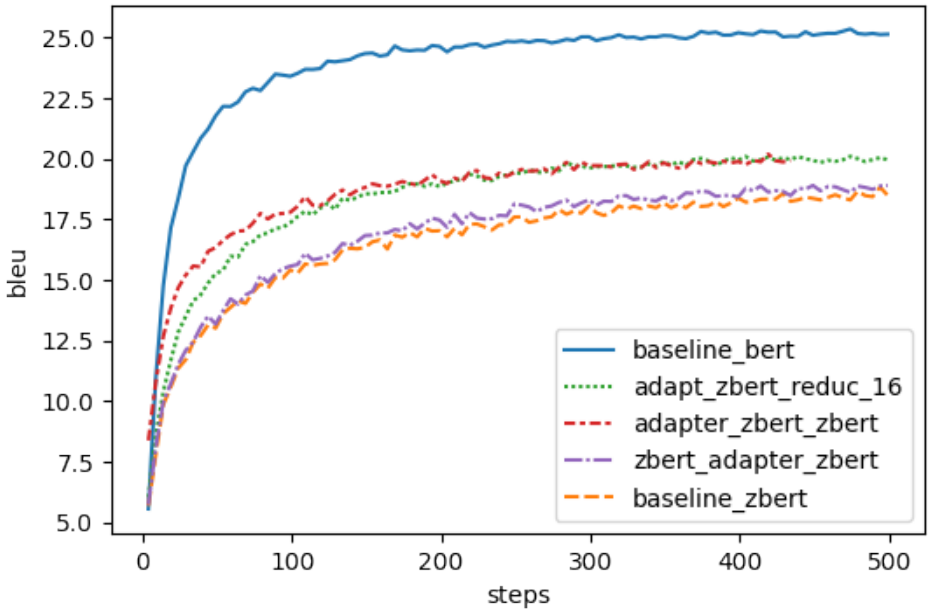

Figure 5.8: Comparison between baseline BERT model, baseline `zbert` model, adapters in both encoder and decoder of `zbert` model (`adapt_zbert_reduc_16`), adapters only in encoder of `zbert` model (`adapter_zbert_zbert`), and adapters only in decoder of `zbert` model (`zbert_adapter_zbert`).

We can see from Figure 5.8 that when we include adapters on both the encoder and decoder, we can outperform the baseline `zbert` in around 2 BLEU points. This shows that, similar to the models that use BERT as the pre-trained model, the adapters can help to improve the performance further, even though some of the information is already missing in the base model.

Furthermore, we can also see that, similar to the BERT model experiments, fine-tuning with adapters only on the encoder side (`adapter_zbert_zbert`) performs

much better than on the decoder side (`zbert_adapter_zbert`). Other than that, we can also see that incorporating adapters only on the encoder side helps the model achieve better performance faster than using adapters on both sides. This further supports our hypothesis that updating the representation on the encoder side is more beneficial. Additionally, we can also see the same behaviour as the original pre-trained BERT that `zbert_adapter_zbert` performance is close to the baseline model (`baseline_zbert`), where we are only fine-tuning the cross-attention and output layers. This could mean that fine-tuning the decoder may not be enough to achieve better performance when the representation from the source side is unchanged.

5.3.2 Model Down-Scaling

Experiment Setup and Motivation

This experiment is the follow-up from `zbert`, where we zeroed out half of the elements in the matrices. More specifically, we are completely removing those elements from the matrix instead of just zeroing out the elements. The way we do this is similar to the one we do on `zsbert`. We remove the matrix elements on every even column and row in the transformer body and the embedding. We again do the weights processing offline before using it as the pre-trained model. For the rest of this writing, we refer to this setup as `zsbert`.

Furthermore, we also follow a similar setup as in Section 5.2.2 where we experiment with the position of the adapters. The goal of this experiment is to understand the behaviour of this model compared to the baseline as well as `zbert`.

Comparison with BERT Baseline and `zbert`

We begin by comparing `zsbert` with the BERT baseline. We see in Figure 5.9 that the performance degrades by more than 10 BLEU points. This is also significantly worse than `zbert`, where we only lose 7.5 BLEU points. Our initial hypothesis was

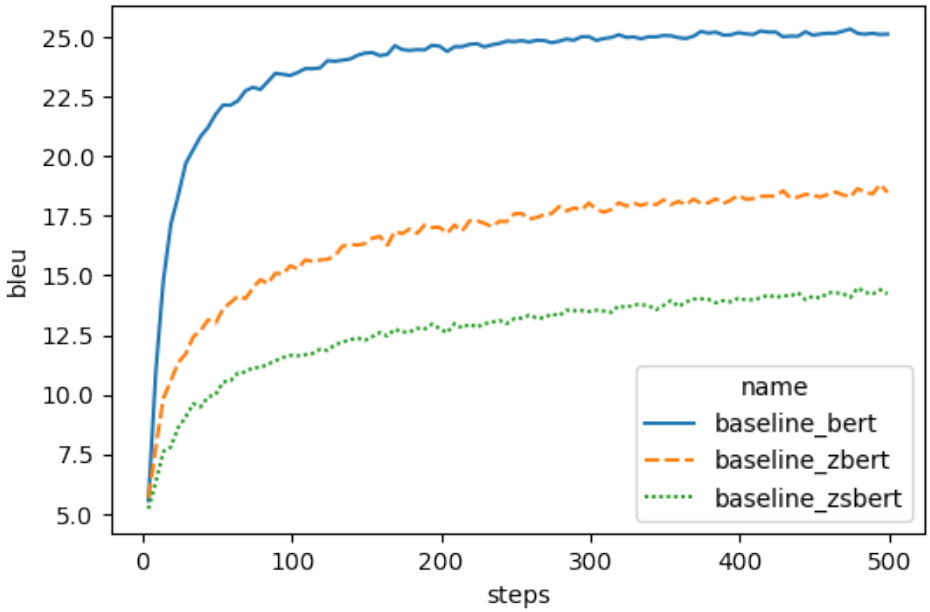

Figure 5.9: Comparison between baseline BERT model and baseline `zsbert` model.

that `zsbert`'s performance should be comparable to `zbert` as we are fundamentally performing the same reduction technique. After some investigation, we realized that removing the weights from the network and zeroing out the matrices are not directly comparable because we also need to consider the computation of layer normalization, which highly depends on the matrix dimension. We found this discrepancy by performing a manual evaluation where we used an arbitrary vector as the input to the network and monitored the output in each of the network layers. We found a slight difference in the layer's output between the zeroed and completely removed weights. Even though the difference is minimal, the output discrepancy gets propagated to the top layers, causing the final network output to differ significantly.

Next, we study the interplay of model down-scaling and adapters. We can see in Figure 5.10 that `zsbert` with a 16 ratio adapters manages to improve the performance up to 6 BLEU points compared to `zbert` without adapters. This shows that adapters can still improve the model's performance even when some weights are missing. Furthermore, despite still showing difficulties in reaching the baseline

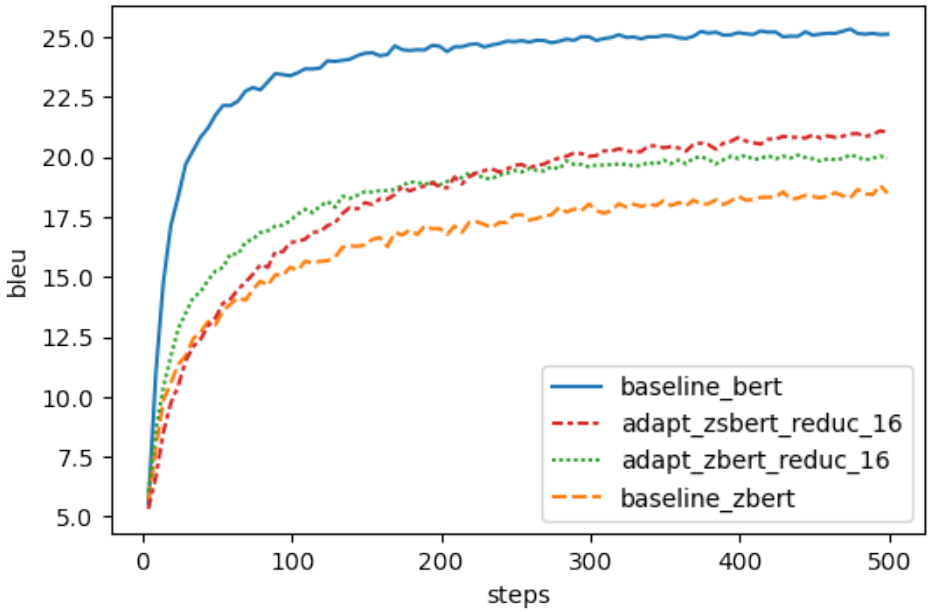

Figure 5.10: Comparison between baseline BERT model, baseline `zbert`, baseline `zsbert` and adapters `zsbert` models.

performance, Figure 5.11 shows that we can still improve the model's performance by reducing the adapters reduction ratio. We can see that the model with the lowest reduction ratio (1) manages to close the performance gap significantly with the baseline model. This is one of our prominent results because we can see from Table 5.1 that the total number of weights (including adapters) required to fine-tune the model is significantly lower than the original BERT model.

Name	# Trainable Variables	# Untrainable Variables	# Total Variables	Percentage Trainable
Ratio 16	7.736.826	95.143.296	102.880.122	7.5%
Ratio 8	8.179.770	95.143.296	103.323.066	7.9%
Ratio 4	9.065.658	95.143.296	104.208.954	8.7%
Ratio 2	10.837.434	95.143.296	105.980.730	10.2%
Ratio 1	14.380.986	95.143.296	109.524.282	13.1%

Table 5.1: Total trainable variables in `zsbert` with adapters on different ratio vs normal BERT model.

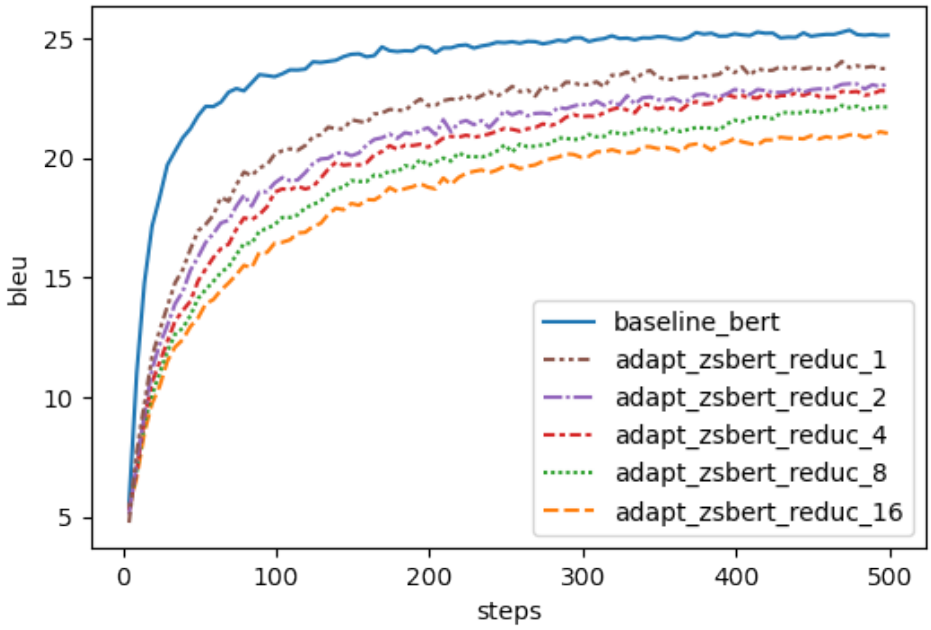

Figure 5.11: Comparison between baseline BERT model and different reduction ratio of `zsbert` models.

Adapters Position

From Figure 5.12, similar to the `zbert` experiments, we can see similar behaviour where models fine-tuned with adapters outperform the baseline `zsbert` and `zbert` models. However, compared to `zbert` experiments, we notice a bigger improvement in `zsbert`'s final performance. In `zbert`, the difference between baseline and adapters is within 5 BLEU points. On the other hand, in `zsbert`, we see the improvement is within 8 BLEU points. This result is particularly interesting for us as we expect the difference to be similar to `zbert`. We recall from 5.3.2 that this is due to the numerical error from the layer normalization, i.e. a difference in constant used to perform vector normalization in the layer normalization. In other words, we observed a worse performance in `zbert` than we got for `zsbert` with adapters.

We deep-dive further in Figure 5.13 to show the comparison between adapters in `zbert` and `zsbert`. We use a reduction ratio of 16 to compare the adapter's performance between these two setups. We notice a leap in the final performance

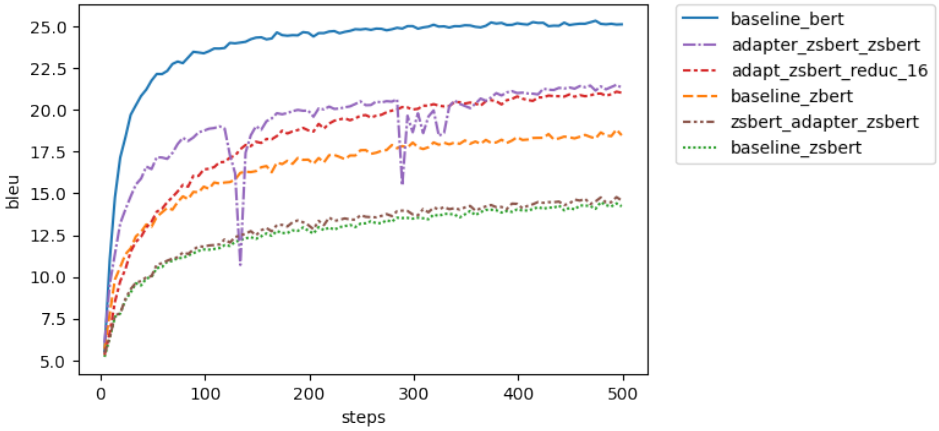

Figure 5.12: Comparison between baseline BERT model, baseline `zsbert` model, adapters in both encoder and decoder of `zsbert` model (`adapt_zsbert_reduc_16`), adapters only in encoder of `zsbert` model (`adapter_zsbert_zsbert`), and adapters only in decoder of `zsbert` model (`zsbert_adapter_zsbert`).

when we compare the adapter model with an equal reduction ratio (16) between `zbert` and `zsbert`. We can see that initially `zsbert` performs worse than `zbert`. After some steps, we can see the performance in `zbert` starting to stall but not in `zsbert`. We hypothesize that this relates to a similar reason we stated in the original BERT model, where we could not see any improvement when increasing the reduction ratio. It is possible that when we reduce the original pre-trained model's size, the adapters adjust the flow of information within the network and better replace the missing information with new knowledge that is more important for solving the task. Another possibility is simply because `zbert` is a "heavier model" than `zsbert` as it contains more parameters and thus has a harder time for the adapters to recover.

We see a similar behaviour as in `zbert` experiments concerning the position of the adapters. In Figure 5.12, the benefit of incorporating adapters on the encoder side is apparent and outperforms the decoder counterpart. We can also see a similar behaviour where the model's performance with adapters on the encoder eventually outperforms the model with adapters on both sides. Furthermore, we also see a similar behaviour as in `zbert` for models with adapters in the decoder only where

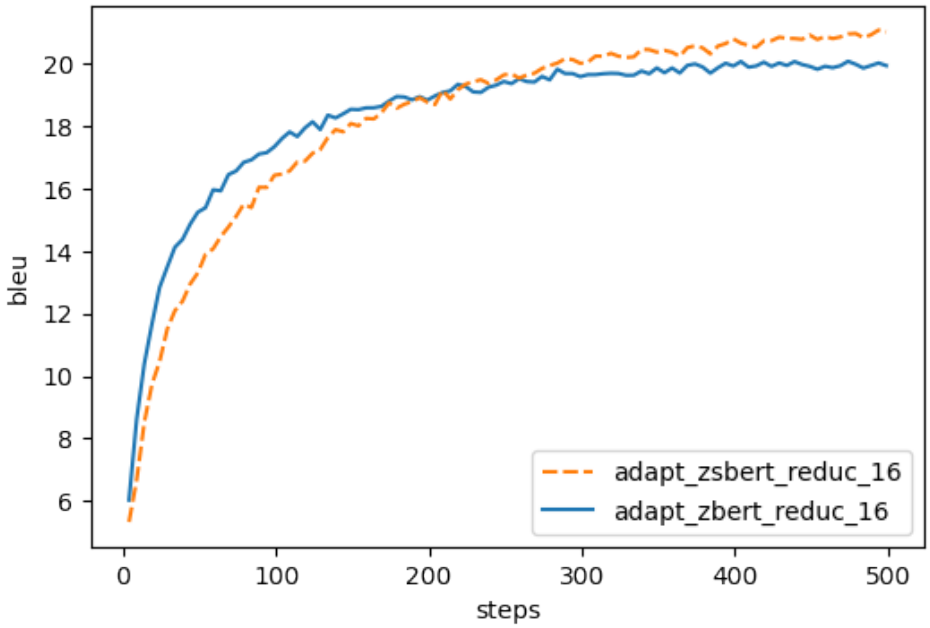

Figure 5.13: Comparison adapters performance in `zsbert` and `zbert`. Both are using reduction ratio 16 and the adapters are placed on encoder and decoder.

the performance is very close to the baseline and not improving as much as on the encoder side. We hypothesize that the same reason as we have stated in `zbert` could apply in `zsbert` as well. Essentially, we need to modify the representation on the encoder side in order to achieve better performance.

Conclusion

5.4 Summary

This thesis has explored various ways to utilize BERT with adapters in machine translation. We start by understanding the impact of pre-training data in different domains and the contribution of different volumes in the pre-training data. We continue the study by leveraging different techniques to understand the impact of the pre-trained models by shuffling the pre-trained BERT weights and using randomly set weights. We then conduct the experiments to understand the importance of adapters in either the encoder or the decoder by showing the performance of adapters when they are removed from either of the components. Finally, we perform reduction experiments where we reduce the size of BERT by manually removing the weights either by zeroing out some of the values in the matrices or completely deleting them.

The experiments in Chapter 4 show that with the proper initialization, adapters can help achieve better performance than training the models from scratch while training far fewer weights than the original model. We further show that even with random fixed weights in the main part of the model, the adapters and cross-attention can recover and achieve performance similar to one of the baseline models.

In the subsequent experiments in Chapter 5, we find that fine-tuning adapters on the encoder side is more important than in the decoder. We also see a similar behaviour when we use the original BERT weights only on the encoder or

the decoder and fixed random weights on the other part. Interestingly, when the adapters were injected only to the decoder, with the encoder pre-trained or random, the performance dropped to zero. In other words, a fitting encoder is critical.

We further studied the behaviour of adapters when we tried to down-scale the pre-trained model size. In our experiments, we found that a model with just half of the weights, such as our `zsbert`, can closely match the performance of the baseline model, the model that uses BERT in both of the encoder and the decoder and only fine-tuning the cross-attention and output layers. Finally, we also observe that we can increase further the effectiveness of adapters in `zsbert` by only incorporating adapters on the encoder side.

We see two practical applications from our findings:

- Initializing just the encoder with the pre-trained weights such as BERT (with a fixed random decoder) and fine-tuning with adapters could be helpful when targeting low-resource languages. The random decoder is created trivially and no large target side monolingual corpus is needed.

- Reducing the pre-trained BERT to half its size and fine-tuning with adapters provide useful GPU memory savings while keeping a similar performance as the baseline model.

In summary, our experiments show the potential of adapters in the machine translation setup. We understand from the experiments that fine-tuning adapters with randomly set weights in the base pre-trained network can achieve similar results as training the entire transformer model with BERT configuration. Furthermore, the down-scaling experiments also show that with a random weight reduction technique, we can reduce the size of BERT and achieve similar performance as the BERT model that was fine-tuned by only modifying the cross-attention layer.

5.5 Future Works

In this thesis, we have seen the behaviour of adapters during fine-tuning on various scenarios where we modify the pre-trained models to the point where we were using a completely random value. It is interesting for us to see that the final performance of the model could sometimes achieve comparable quality as if we train the whole BERT model from scratch. Further experiments would be interesting to solidify the results found in this thesis. We propose experiments with various model architectures to see whether we can still see the same behaviour. The goal of the experiments is to measure any correlations between the adapters and the architecture chosen for the base model. Until recently, most works in adapters only focus on transformer-based models. It would be interesting to perform similar experiments with LSTM-based models.

There are some works such as [6, 66] where they can identify the location of certain features and pieces of information within the pre-trained models. In this thesis, we have shown that when we remove arbitrary weights from BERT and later perform the fine-tuning with adapters, the model can perform similarly to the model that uses the whole BERT weights and only modify the cross-attention and output layers. This experiment shows that some lost information can be recovered in the adapters during fine-tuning as long as the adapters have enough capacity. It would be interesting if we could find a way to store a particular group of features by training the models modularly with adapters. [55] has shown that by incorporating adapters during the pre-training and using them as a module that stores the knowledge for each language, they can maintain a constant size of the model while increasing the number of languages that they feed to the model. It would be interesting to perform experiments where the adapters can store more granular linguistics features such as syntax and morphology level features.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] A. Bapna, M. X. Chen, O. Firat, Y. Cao, and Y. Wu. Training deeper neural machine translation models with transparent attention. In *EMNLP*, 2018.

[3] A. Bapna and O. Firat. 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, Nov. 2019. Association for Computational Linguistics.

[4] A. V. M. Barone, B. Haddow, U. Germann, and R. Sennrich. Regularization techniques for fine-tuning in neural machine translation. In *EMNLP*, 2017.

[5] L. Barrault, O. Bojar, M. R. Costa-jussà, C. Federmann, M. Fishel, Y. Graham, B. Haddow, M. Huck, P. Koehn, S. Malmasi, C. Monz, M. Müller, S. Pal, M. Post, and M. Zampieri. Findings of the 2019 conference on machine translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy, Aug. 2019. Association for Computational Linguistics.

[6] M. Ben Noach and Y. Goldberg. Compressing pre-trained language models by matrix decomposition. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 884–889, Suzhou, China, Dec. 2020. Association for Computational Linguistics.

[7] O. Bojar, C. Buck, C. Callison-Burch, C. Federmann, B. Haddow, P. Koehn, C. Monz, M. Post, R. Soricut, and L. Specia. Findings of the 2013 Workshop on Statistical Machine Translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 1–44, Sofia, Bulgaria, Aug. 2013. Association for Computational Linguistics.

[8] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, and A. Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

[9] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, S. Huang, M. Huck, P. Koehn, Q. Liu, V. Logacheva, C. Monz, M. Negri, M. Post, R. Rubino, L. Specia, and M. Turchi. Findings of the 2017 conference on machine translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics.

[10] O. Bojar, R. Chatterjee, C. Federmann, Y. Graham, B. Haddow, M. Huck, A. Jimeno Yepes, P. Koehn, V. Logacheva, C. Monz, M. Negri, A. Névéol, M. Neves, M. Popel, M. Post, R. Rubino, C. Scarton, L. Specia, M. Turchi, K. Verspoor, and M. Zampieri. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany, Aug. 2016. Association for Computational Linguistics.

[11] O. Bojar, R. Chatterjee, C. Federmann, B. Haddow, C. Hokamp, M. Huck, V. Logacheva, and P. Pecina. Proceedings of the tenth workshop on statistical machine translation, wmt@emnlp 2015, 17-18 september 2015, lisbon, portugal. In *WMT@EMNLP*, 2015.

[12] O. Bojar, R. Chatterjee, C. Federmann, B. Haddow, M. Huck, C. Hokamp, P. Koehn, V. Logacheva, C. Monz, M. Negri, M. Post, C. Scarton, L. Specia, and M. Turchi. Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 1–46, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics.

[13] O. Bojar, C. Federmann, M. Fishel, Y. Graham, B. Haddow, P. Koehn, and C. Monz. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels, Oct. 2018. Association for Computational Linguistics.

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

[15] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. 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. 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. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[16] C. Callison-Burch, C. Fordyce, P. Koehn, C. Monz, and J. Schroeder. (meta-) evaluation of machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 136–158, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[17] C. Callison-Burch, C. Fordyce, P. Koehn, C. Monz, and J. Schroeder. Further meta-evaluation of machine translation. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 70–106, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[18] C. Callison-Burch, P. Koehn, C. Monz, K. Peterson, M. Przybocki, and O. Zaidan. Findings of the 2010 joint workshop on statistical machine translation and metrics for machine translation. In *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*, pages 17–53, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[19] C. Callison-Burch, P. Koehn, C. Monz, M. Post, R. Soricut, and L. Specia. Findings of the 2012 workshop on statistical machine translation. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 10–51, Montréal, Canada, June 2012. Association for Computational Linguistics.

[20] C. Callison-Burch, P. Koehn, C. Monz, and J. Schroeder. Findings of the 2009 Workshop on Statistical Machine Translation. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 1–28, Athens, Greece, Mar. 2009. Association for Computational Linguistics.

[21] C. Callison-Burch, P. Koehn, C. Monz, and O. Zaidan. Findings of the 2011 workshop on statistical machine translation. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 22–64, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[22] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico. Report on the 11th iwslt evaluation campaign. In *IWSLT*, 2014.

[23] Y.-C. Chen, Z. Gan, Y. Cheng, J. Liu, and J. Liu. Distilling the knowledge of bert for text generation. 2019.

[24] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of*

SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, pages 103–111, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[25] C. Chu, R. Dabre, and S. Kurohashi. An empirical comparison of domain adaptation methods for neural machine translation. In *ACL*, 2017.

[26] C. Chu and R. Wang. A survey of domain adaptation for neural machine translation. In *COLING*, 2018.

[27] A. CONNEAU and G. Lample. Cross-lingual language model pretraining. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[28] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[30] W. B. Dolan and C. Brockett. Automatically constructing a corpus of sentential paraphrases. In *IJCNLP*, 2005.

[31] M. Freitag and Y. Al-Onaizan. Fast domain adaptation for neural machine translation. *ArXiv*, abs/1612.06897, 2016.

[32] J. Gao and M. Zhang. Improving language model size reduction using better pruning criteria. In *ACL*, 2002.

[33] J. Guo, Z. Zhang, L. Xu, B. Chen, and E. Chen. Adaptive adapters: An efficient way to incorporate bert into neural machine translation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:1740–1751, 2021.

[34] W. Han, B. Pang, and Y. N. Wu. Robust transfer learning with pretrained language models through adapters. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 854–861, Online, Aug. 2021. Association for Computational Linguistics.

[35] A. S. Hildebrand, M. Eck, S. Vogel, and A. H. Waibel. Adaptation of the translation model for statistical machine translation based on information retrieval. In *EAMT*, 2005.

[36] 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. PMLR, 2019.

[37] J. Howard and S. Ruder. 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)*, pages 328–339, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[38] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.

[39] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *ArXiv*, abs/1811.06965, 2019.

[40] D. Johnson. Seq2seq (sequence to sequence) model with pytorch, May 2022.

[41] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA, Oct. 2013. Association for Computational Linguistics.

[42] P. Koehn and R. Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver, Aug. 2017. Association for Computational Linguistics.

[43] P. Koehn and C. Monz. Manual and automatic evaluation of machine translation between European languages. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 102–121, New York City, June 2006. Association for Computational Linguistics.

[44] O. Kuchaiev and B. Ginsburg. Factorization tricks for LSTM networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[45] C. Lee, K. Cho, and W. Kang. Mixout: Effective regularization to finetune large-scale pretrained language models. *ArXiv*, abs/1909.11299, 2020.

[46] T. Lee, M.-J. Lee, T. G. Kang, S. Jung, M. Kwon, Y. Hong, J. Lee, K.-G. Woo, H.-G. Kim, J. Jeong, et al. Adaptable multi-domain language model for transformer asr. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7358–7362. IEEE, 2021.

[47] L. Liu, X. Liu, J. Gao, W. Chen, and J. Han. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763, Online, Nov. 2020. Association for Computational Linguistics.

[48] M.-T. Luong and C. D. Manning. Stanford neural machine translation systems for spoken language domains. In *IWSLT*, 2015.

[49] D. Macháček. Enriching neural mt through multi-task training. 2018.

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

[51] M. Mosbach, M. Andriushchenko, and D. Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *ArXiv*, abs/2006.04884, 2021.

[52] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[53] D. Park and C. W. Ahn. Self-supervised contextual data augmentation for natural language processing. *Symmetry*, 11:1393, 2019.

[54] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *NAACL*, 2018.

[55] 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.

[56] 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, Apr. 2021. Association for Computational Linguistics.

[57] J. Pfeiffer, A. Rücklé, C. Poth, A. Kamath, I. Vulić, S. Ruder, K. Cho, and I. Gurevych. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online, Oct. 2020. Association for Computational Linguistics.

[58] 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, Nov. 2020. Association for Computational Linguistics.

[59] 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, 2020.

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

[61] A. Radford and K. Narasimhan. Improving language understanding by generative pre-training. 2018.

[62] A. Radford and K. Narasimhan. Improving language understanding by generative pre-training. 2018.

[63] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[64] P. Rajpurkar, R. Jia, and P. Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[65] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016.

[66] S. Ravfogel, Y. Elazar, J. Goldberger, and Y. Goldberg. Unsupervised distillation of syntactic information from contextualized word representations. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 91–106, Online, Nov. 2020. Association for Computational Linguistics.

[67] A. Rücklé, G. Geigle, M. Glockner, T. Beck, J. Pfeiffer, N. Reimers, and I. Gurevych. Adapterdrop: On the efficiency of adapters in transformers. *ArXiv*, abs/2010.11918, 2020.

[68] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, 2019.

[69] E. T. K. Sang and F. D. Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *CoNLL*, 2003.

[70] M. Schuster and K. Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[71] R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. *ArXiv*, abs/1511.06709, 2016.

[72] R. Sennrich, B. Haddow, and A. Birch. 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, Aug. 2016. Association for Computational Linguistics.

[73] C. Servan, J. M. Crego, and J. Senellart. Domain specialization: a post-training domain adaptation for neural machine translation. *ArXiv*, abs/1612.06141, 2016.

[74] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[75] N. M. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman. Mesh-tensorflow: Deep learning for supercomputers. *ArXiv*, abs/1811.02084, 2018.

[76] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[77] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, 2010.

[78] 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.

[79] L. Voita. Sequence to sequence (seq2seq) and attention, Apr 2022.

[80] A. Wang, J. Hula, P. Xia, R. Pappagari, R. T. McCoy, R. Patel, N. Kim, I. Tenney, Y. Huang, K. Yu, S. Jin, B. Chen, B. V. Durme, E. Grave, E. Pavlick, and S. R. Bowman. Can you tell me how to get past sesame street? sentence-level pretraining beyond language modeling. In *ACL*, 2019.

[81] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. 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*, pages 353–355, Brussels, Belgium, Nov. 2018. Association for Computational Linguistics.

[82] R. Weng, H. Yu, S. Huang, S. Cheng, and W. Luo. Acquiring knowledge from pre-trained model to neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9266–9273, 2020.

[83] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*, 2018.

[84] G. I. Winata, G. Wang, C. Xiong, and S. Hoi. Adapt-and-adjust: Overcoming the long-tail problem of multilingual speech recognition, 2020.

[85] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[86] J. Yang, M. Wang, H. Zhou, C. Zhao, W. Zhang, Y. Yu, and L. Li. Towards making the most of bert in neural machine translation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 9378–9385, 2020.

[87] D. Yogatama, C. de Masson d'Autume, J. T. Connor, T. Kociský, M. Chrzanowski, L. Kong, A. Lazaridou, W. Ling, L. Yu, C. Dyer, and P. Blunsom. Learning and evaluating general linguistic intelligence. *ArXiv*, abs/1901.11373, 2019.

[88] K. W. Zhang and S. R. Bowman. Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In *BlackboxNLP@EMNLP*, 2018.

[89] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. gang Zhou, H. Li, and T.-Y. Liu. Incorporating bert into neural machine translation. *ArXiv*, abs/2002.06823, 2020.