The Kronecker Product for Efficient Natural Language Processing

Ali Edalati
Department of Electrical and Computer Engineering
McGill University, Montreal
April, 2023

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

To the victims of Flight PS752

Abstract

The development of Transformers has heralded a major breakthrough in deep learning and its application. For example, modern Natural Language Processing (NLP) systems have achieved marvelous results by utilizing Pre-trained Language Models (PLMs) as a backbone in their structure. Most PLMs are huge models, developed by stacking several layers of Transformers. These models are pre-trained on a massive dataset to learn a general knowledge of natural languages.

However, there are some major challenges associated with the superior performance of Transformer-based PLMs. Firstly, the size of these models is so huge that it is impossible to apply them on edge devices with limited memory, energy, and computation capacity. Secondly, fine-tuning PLMs on downstream datasets faces two main challenges that are crucial to address considering the ever-growing size of PLMs. First, it has become more time-consuming. Second, fine-tuning requires devices with large memories that enable training as well as storing the fine-tuned weights for each dataset.

To mitigate the mentioned issues, there has been a growing volume of research focused on two directions. In the first direction, the goal is compressing a model by reducing its number of parameters while maintaining the performance and expressiveness of the model. This direction is named Model Compression. The second direction is Parameter Efficient Tuning (PET), which aims to fine-tune a model by reducing the number of trainable parameters, which usually results in accelerated training and reduced memory consumption.

In this work, we use the Kronecker product in the two mentioned directions separately. First, a Model Compression technique is developed, called Kronecker Decomposition, that uses the Kronecker product as a backbone. This technique is applied to Generative Pre-trained Transformer-2 (GPT-2); a PLM with few existing compressed versions. We use Kronecker Decomposition to decompose GPT-2 into Kronecker Generative Pre-trained Transformer-2 (KnGPT2). After decomposition that initializes KnGPT2, a limited pre-training is required to retrieve the performance of the compressed model. Then, KnGPT2 is adapted to downstream tasks by fine-tuning, similar to the uncompressed models. Note that a proposed Intermediate Layer Knowledge Distillation (ILKD) technique is used during both the pre-training and fine-tuning stages to minimize the performance drop of Kronecker Decomposition.

Second, we aimed to improve the performance of Adapter-based PET techniques by incorporating the Kronecker product. Adapter-based methods are considered one of the most successful PET techniques. Adapters are often low-rank modules that are inserted into a model. While the model is frozen during fine-tuning, the adapters are tuned to simulate fine-tuning of the model. Nonetheless, the low-rank nature of adapters limits their representation power. In this work, we developed Kronecker Adapter (KronA) by incorporating the Kronecker product in adapters to address the mentioned problem.

Finally, by providing empirical results, we show that our proposed KnGPT2 significantly outperforms DistilGPT2 as a baseline on both language modeling and the General Language Understanding Evaluation (GLUE) tasks. Also, we applied the proposed Kronecker-based adapters as well as other novel PET techniques to Text-to-Text Transfer Transformer (T5) to evaluate them on the GLUE benchmark. The results support our hypothesis about the high potential of the Kronecker product in PET since our proposed Kronecker-based modules outperformed the state-of-the-art baselines.

Abrégé

Le développement de Transformateurs a marqué une révolution dans l'apprentissage profond et ses applications. Par exemple, les systèmes modernes de Traitement Traitement Automatique du Langage Naturel (TALN) ont obtenu des résultats merveilleux en utilisant des Modèles de Langage Pré-entraînés (MLP) comme une base de leur structure. La plupart de ces MLP sont immenses, sont développés en empilant plusieurs couches de Transformateurs et sont pré-entraînés sur un ensemble de données massif pour apprendre une connaissance générale des langues naturelles.

Néanmoins, des défis majeurs sont associés à la performance supérieure des modèles pré-entrainés basés sur les Transformateurs. Premièrement, la taille de ces modèles est si énorme qu'il est quasiment impossible de les déployer sur des appareils périphériques dotés d'une capacité de mémoire, d'énergie et de calcul limitée. Deuxièmement, le réglage fin de ces MLP sur les ensembles de données en aval est confronté à deux défis cruciaux: Tout d'abord, cette tâche peut prendre beaucoup de temps à réaliser. Aussi, le réglage fin nécessite des appareils ayant de grandes mémoires qui permettent l'entrainement ainsi que le stockage des poids affinés pour chaque ensemble de données.

Pour atténuer ces problèmes mentionnés, il y a eu un grand volume de recherches axées sur deux directions. Dans le premier sens, le but est de compresser un modèle en réduisant son nombre de paramètres tout en conservant les performances et l'expressivité du modèle. Cette direction est connue comme Compression du Modèle. La deuxième direction est le Tuning de Paramètres Efficace (TPE) Ceci vise à affiner un modèle en réduisant le nombre de paramètres entraînables, ce qui se traduit généralement par un entraînement accéléré et une consommation de mémoire réduite.

Nous utilisons le produit Kronecker dans les deux directions mentionnées auparavant. D'abord, une technique de Compression de Modèle est développée, appelée Décomposition de Kronecker. Cette technique utilise comme épine dorsale le produit de Kronecker. Cette technique est appliquée au GPT-2 qui est un modèle pré-entraîné avec peu de versions compressées existantes. Particulièrement, nous utilisons la Décomposition de Kronecker pour décomposer GPT-2 en KnGPT2. Après décomposition, un préapprentissage léger est nécessaire pour atteindre une bonne performance du modèle compressé. Ensuite, KnGPT2 est adapté aux tâches en aval par un réglage fin, similaire aux modèles non compressés. Notez qu'une technique proposée de Distillation de Connaissances par les Couches Intermédiaires (DCCI) est utilisée pendant les étapes de pré-entrainement et de réglage fin pour minimiser la baisse de performance de la Décomposition de Kronecker.

Ensuite, nous avons amélioré les performances des techniques TPE basées sur des adaptateurs en incorporant le produit Kronecker. Les méthodes basées sur les adaptateurs sont considérées comme l'une des techniques les plus réussies. Les adaptateurs sont souvent des modules de bas rang qui sont insérés dans un modèle. Pendant que le modèle est gelé pendant le réglage fin, les adaptateurs sont réglés pour simuler le réglage fin du modèle. Cependant, la nature de bas rang des adaptateurs limite leur pouvoir de représentation. Dans ce travail, nous avons développé KronA en incorporant le produit Kronecker dans des adaptateurs pour résoudre le problème mentionné.

Enfin, à travers des résultats empiriques, nous montrons que notre KnGPT2 proposé surpasse de manière significative DistilGPT2 comme référence à la fois sur la modélisation du langage et sur les tâches GLUE. En outre, nous avons appliqué les adaptateurs proposés à base de Kronecker ainsi que d'autres nouvelles techniques TPE à T5 pour les évaluer sur le benchmark GLUE. Les résultats confirment notre hypothèse sur le potentiel élevé du produit Kronecker en TPE puisque nos modules basés sur Kronecker proposés ont surpassé les performances des modèles de l'état de l'art.

Acknowledgements

First and foremost, I would like to express my most profound appreciation and gratitude for my supervisor, Prof. James J. Clark, whose guidance, support, patience, and encouragement have been invaluable throughout this research project. He guides all of his students with heart. I learned a lot from his deep and vast knowledge. Also, I will never forget his incredible compassion, understanding, and generosity.

I am deeply in debt to Prof. Warren J. Gross, Prof. Brett H. Meyer, and Prof. James J. Clark, co-directors of McGill Edge Intelligence Laboratory (MEIL), for creating this great lab that raises opportunities for top-notch research.

From the bottom of my heart, I would like to say a special thank you to Dr. Mehdi Rezagholizadeh, my former supervisor during my internship at Huawei Noah's Ark Laboratory. Without his support and guidance, performing this work was not possible. Also, I am highly grateful to Dr. Vahid Partovi Nia, my latter supervisor at Huawei Noah's Ark Laboratory, who had an important role in this research.

This endeavor would not have been possible without the support of Huawei Noah's Ark Laboratory. Also, special thanks to all of my colleagues, specifically Dr. Marzieh Tahaei, Ahmad Rashid, and Dr. Ivan Kobyzev.

I would like to acknowledge my lab mates at MEIL and Centre for Intelligent Machines (CIM) at McGill. I wish you all the best. Specifically, Ibtihel Amara, who helped me with writing the French abstract.

Words cannot express my gratitude to my beloved parents and sister, whose consistent support has been the main motivation in my life.

Table of Contents

List of Figures

List of Tables

Acronyms

KronA$^{\mathbf{B}}$ Kronecker Adapter for Blocks. x, xi, xv, xviii, 4, 5, 50, 51, 60, 61, 64, 68, 69, 74–76, 78, 82

KronA$^{\mathbf{B}}_{\mathbf{res}}$ Kronecker Adapter for Blocks with Residual Connection. xi, xv, xviii, xix, 4, 5, 51, 52, 60, 61, 65, 69, 70, 74–76, 78

KronA$^{\mathbf{M}}_{\mathbf{sig\text{-}res}}$ Kronecker Adapter for Modules with Sigmoid Residual Connection. xix, 70

Acc Accuracy. xi, 56, 61

ACL Annual Meeting of the Association for Computational Linguistics. 5

AX Diagnostics Main. 54

BART Bidirectional and Auto-Regressive Transformer. ix, xiii, xvi, 21–23

BERT Bidirectional Encoder Representations from Transformers. ix, xiii, xiv, xvi, 2, 18–21, 29, 32

C4 Colossal Clean Crawled Corpus. 23

CE Cross Entropy. xviii, 46, 47, 67

CIM Centre for Intelligent Machines. viii

CLM Casual Language Modeling. ix, xiii, 4, 14, 20, 23, 26, 32, 54, 72

CoLA Corpus of Linguistic Acceptability. 54, 56, 57, 63–65, 67–69, 73, 74, 76

DCCI Distillation de Connaissances par les Couches Intermédiaires. vii

FFLM Feed Forward Language Model. xiii, 11–13

FFN Feed-Forward Network. xiv, 16, 17, 35, 37–39, 50, 51, 59, 61, 68, 69

FLOPS Floating Point Operations Per Second. 41, 79

FN False Negative. 57, 58

FP False Positive. 57, 58

FT Full Fine-Tuning. x, xvii, 1, 2, 33, 61, 62, 74–77, 80, 82

GB Giga Byte. 71

GELU Gaussian Error Linear Unit. 36, 68, 69

GLUE General Language Understanding Evaluation. v, vii, xi, xii, xix, 4, 5, 19, 54, 55, 59, 60, 62, 63, 68–70, 72–74, 76, 81, 82

GPT Generative Pre-trained Transformer. v, vii, ix, x, xiii, xv, xvi, 2–6, 19–21, 24–27, 32, 40, 42, 45, 46, 59, 60, 71–73, 77–79, 81

GPU Graphics Processing Unit. 59

ILKD Intermediate Layer Knowledge Distillation. v, x, xi, xvi, 3, 45, 46, 60, 67, 72, 73, 77, 82

IoT Internet of Things. 31, 32

K-L Kullback–Leibler. xviii, 46, 67

KD Knowledge Distillation. x, xiv, xvii, 2, 3, 26, 28–32, 60, 73, 81

KnGPT2 Kronecker Generative Pre-trained Transformer-2. v, vii, xi, xv, xviii, 3, 4, 45, 46, 59, 60, 66, 67, 71–73, 81, 82

KronA Kronecker Adapter. v, vii, x, xi, xv–xviii, 4, 5, 48–50, 61, 64, 68, 74–78, 82

KU Kaming-Uniform. 50, 68

LM Language Model. ix, xiv, 2, 9–12, 14, 18, 24–27, 31–33, 44, 56, 72, 77

LoRA Low Rank Adaption. x, xiv–xvii, 3, 37–39, 48–50, 60, 61, 63, 74–76, 82

LSTM Long-Short Term Memory. xiii, 13, 14, 16

LTR Left To Right. 14, 21, 26

mBART Multi-Lingual Bidirectional and Auto-Regressive Transformer. 22

MCC Matthews's Correlation Coefficient. xi, 57, 58

MEIL McGill Edge Intelligence Laboratory. viii

MHA Multi-Head Attention. xiii, 16–18, 35, 39

MLM Masked Language Modeling. ix, xiii, 14, 15, 19, 32

MLP Modèles de Langage Pré-entraînés. vi

MLP Multi Layer Perceptron. 34

MNLI Multi-Genre Natural Language Inferenc. xvii, 54–56, 61, 63–69, 73, 74, 76

MRPC Microsoft Research Paraphrase Corpus. 54, 56, 57, 63–65, 67–69, 73, 74, 76

MSE Mean Squared Error. xviii, 46, 67

NLP Natural Language Processing. iv, 1, 9, 10, 27, 32, 54, 81

NLU Natural Language Understanding. 19, 20, 22, 25

NNLM Neural Network Language Model. ix, 11

NSP Next Sentence Prediction. 19

OWT Open Web Text. xi, 3, 26, 53, 60, 72

PA Parallel Adapter. x, xiv, xv, xvii, 3, 36–39, 50, 51, 60, 61, 63, 74–76, 82

PCC Pearson's Correlation Coefficient. xi, 58

PET Parameter Efficient Tuning. iv, v, x–xii, xvii, xix, 2–5, 9, 23, 32–34, 36, 41, 53, 55, 60, 61, 68, 71, 74–77, 79, 80, 82

PLM Pre-trained Language Model. iv, v, 1, 2, 4, 23, 37–39, 48, 49, 68, 77, 80, 81

PPL Perplexity. xi, xix, 4, 56, 66, 72

PS Parallel + Scale. 69

QNLI Question-Answering Natural Language Inference. xviii, 54–56, 63–65, 67–69, 73, 74, 76

QQP Quora Question Pairs. 55, 56, 63–65, 67–69, 73, 74, 76

ReLU Rectified Linear Unit. 37, 68, 69

RNN Recurrent Neural Network. 12, 13, 16

RNNLM Recurrent Neural Network Language Model. xiii, 12–14

RTE Recognizing Textual Entailment. 55, 56, 63–65, 67–69, 73, 74, 76

SCC Spearman's Rank Correlation Coefficient. xi, 58

SiLU Sigmoid Linear Unit. 68, 69

SOTA State-of-the-Art. xix, 1, 5, 34, 36, 74, 80–82

SQuAD Singular Value Decomposition. 54

SST-2 Stanford Sentiment Treebank-2. 55, 56, 63–65, 67–69

STS-B Semantic Textual Similarity Benchmark. xvii, xviii, 55, 56, 58, 63–65, 68, 69, 74, 76

SVD Singular Value Decomposition. xiv, 43

T5 Text-to-Text Transfer Transformer. v, vii, ix, xiv, xvi, 23, 24, 60, 74

TALN Traitement Automatique du Langage Naturel. vi

TN True Negative. 57, 58

TP True Positive. 57, 58

TPE Tuning de Paramètres Efficace. vi, vii

WNLI Winograd Natural Language Inference. 54

Chapter 1

Introduction

Utilizing large Pre-trained Language Models (PLMs) in Natural Language Processing (NLP) systems has enabled achieving State-of-the-Art (SOTA) results [Devlin et al., 2019, Radford et al., 2019, Shoeybi et al., 2019, Yang et al., 2019]. Transformers are the main component in the architecture of most PLMs. The most common paradigm in using PLMs is first pre-training on a huge unlabeled dataset, then fine-tuning on a specific downstream dataset.

One recent problem is that the ever-growing size of PLMs (from hundreds of millions to hundreds of billions of parameters) prevents the deployment on lower-capacity devices with memory, computation, and energy constraints. Examples of these devices are smartphones and smartwatches.

In addition to the mentioned problem, large PLMs need to be adapted to downstream tasks. On the one hand, In-Context Learning methods [Liu et al., 2022, Xie et al., 2022] are a potential solution to adapt a pre-trained model to downstream tasks by providing the model with several examples of a task. However, these methods notably increase the required time, memory, and computations of the inference stage since the model should process all of the provided examples for each inference sample. On the other hand, Full Fine-Tuning (FT) of a pre-trained model on each downstream task usually achieves better results without increasing the time, memory, or computations required for the inference.

In spite of the mentioned benefits, FT of large PLMs has become remarkably time-taking, memory-consuming, and resource-demanding due to their huge size. In addition, a separate checkpoint of the model's tuned weights should be stored for each downstream task, which consumes a tremendous amount of memory [Edalati et al., 2022a].

There is one research direction called Model Compression, which aims to compress large models by decreasing their number of parameters while maintaining their performance. Note that by compressing a model, the required memory for storing the model in both the training and inference phases decreases. Also, the Model Compression techniques may reduce the training or inference time. The main approaches for compression of PLMs are [Edalati et al., 2022b, Tahaei et al., 2022]: Quantization [Prato et al., 2020, Zhang et al., 2020], Pruning [Han et al., 2016], Knowledge Distillation (KD) [Hinton et al., 2015] and Matrix Decomposition [Lioutas et al., 2020, Yu et al., 2017].

Parameter Efficient Tuning (PET) techniques are developed to address the challenges related to fine-tuning large PLMs. These approaches tune a relatively small number of parameters that could be either inserted into or a subset of the original parameters of the pre-trained model. By reducing the number of trainable parameters, the required time and memory for fine-tuning decreases. Also, by storing only the set of trainable parameters (which is significantly smaller compared to the entire model), the tuned checkpoint for each downstream task can be reproduced. Therefore, the need for large memories to store the entire tuned checkpoints is eliminated.

On the one hand, the compression of encoder-based PLMs (mostly Bidirectional Encoder Representations from Transformers (BERT) family) has been sufficiently studied in the literature. TernaryBERT [Zhang et al., 2020], DistilBERT [Sanh et al., 2019], MobileBERT [Sun et al., 2020], MATE-KD [Rashid et al., 2021], Annealing-KD [Jafari et al., 2021], ALP-KD [Passban et al., 2021], TinyBERT [Jiao et al., 2020], and BinaryBERT [Bai et al., 2021] are all example of BERT compression. On the other hand, decoder-based Language Models (LMs) such as Generative Pre-trained Transformer-2 (GPT-2) have a few compressed versions [Li et al., 2021]. DistilGPT2 is a well-known compressed version of GPT-2 which is developed by Knowledge Distillation (KD) from GPT-2 during pre-training on the Open Web Text (OWT) dataset [Edalati et al., 2022b].

Soft Prompt Tuning approaches are one category of PET methods that concatenate trainable vectors to the input of frozen pre-trained models [Lester et al., 2021, Li and Liang, 2021]. Increasing the length of the input sequence, which leads to a notable expansion of inference computations, is considered a disadvantage of these methods. Adapter-based approaches are another category of PET techniques that insert several trainable modules, called adapters, to frozen pre-trained models during the fine-tuning stage. Based on the architecture of an adapter and its position in a model, there are several versions of adapters. For example, Adapter and Parallel Adapter (PA) [He et al., 2022a] insert basic adapters into the Transformer blocks sequentially and in-parallel, respectively. Compacter [Karimi Mahabadi et al., 2021] is a sequential adapter that has a complex architecture composed of a combination of the Kronecker product and normal matrix multiplication. Low Rank Adaption (LoRA) [Hu et al., 2022] is a type of parallel adapter that is applied to the weight matrices instead of blocks. In contrast to other adapters, LoRA does not have a non-linear activation function and it is applied to the linear weight matrices instead of non-linear blocks. Therefore, LoRA adapters can be merged into the pre-trained model after fine-tuning. Consequently, LoRA does not increase the inference latency and memory in contrast to the other adapters.

Our work introduces two methods for Model Compression and PET, respectively. Firstly, we modified the algorithm that is developed by [Tahaei et al., 2022] to compress GPT-2. This method is called Kronecker Decomposition. In our method, the weight matrices of GPT-2's linear layers are decomposed into corresponding Kronecker factors. The decomposition leads to a significant performance drop. Therefore, we perform a limited pre-training on 10% of the DistilGPT2 pre-training data to retrieve the performance. Then, KnGPT2 is fine-tuned on downstream tasks to be compared with the baseline. A developed Intermediate Layer Knowledge Distillation (ILKD) technique is utilized at the pre-training and fine-tuning stages to further improve the performance. The ILKD technique uses the teacher's intermediate layers outputs as well as the last layer output to train the compressed model, KnGPT2. Although we applied this technique to GPT-2, it can be utilized to compress other Transformer-based models as well.

Secondly, we developed adapters that replace the normal matrix multiplication with the Kronecker product to introduce three versions of Kronecker-based adapters. Kronecker Adapter (KronA) that is applied to fine-tune weight matrices and can be merged into the pre-trained model at the inference stage. Therefore, KronA does not increase the inference latency. In addition, we developed Kronecker Adapter for Blocks (KronA^{B}) by inserting the Kronecker adapters in-parallel to pre-trained blocks instead of weight matrices. While KronA^{B} outperforms KronA, it increases the inference latency since KronA^{B} modules cannot be merged into the pre-trained model. We also introduced Kronecker Adapter for Blocks with Residual Connection ($\text{KronA}^{\text{B}}_{\text{res}}$), which benefits from a scaled residual connection to achieve a better performance at the expense of even more latency.

To show the superiority of our proposed techniques, we evaluated KnGPT2 on Casual Language Modeling (CLM) in addition to the General Language Understanding Evaluation (GLUE) benchmark [Wang et al., 2019]. Also, our proposed PET methods are only evaluated on GLUE to study the impact of the Kronecker product.

We summarized our contributions in the following points:

- According to the information we have, this is the first work that uses Kronecker Decomposition to compress GPT-2, which is a large generative PLM.
- We proposed a compression technique that generates KnGPT2. While KnGPT2 significantly outperforms the baseline, DistilGPT2, in terms of the GLUE score and CLM Perplexity, it is pre-trained remarkably faster.
- We developed KronA, an adapter that uses the Kronecker product as a backbone and is utilized to fine-tune the weight matrices of a pre-trained model.

- We introduced KronA^B and KronA$^B_{\text{res}}$, two modified versions of KronA for fine-tuning blocks of a pre-trained model to achieve better results at the expense of an increased inference latency.
- We compared our proposed Kronecker adapters to SOTA baselines in terms of the GLUE score, training time, and inference time to show their advantages.

1.1 Thesis Organization

The remainder of the thesis is organized as follows. In Chapter 2, required preliminaries and related work to the thesis are discussed. Our methodology and solutions for Model Compression and PET are introduced in Chapters 3 and 4, respectively. Details about the experimental setups are provided in Chapter 5. Chapter 6 covers an ablation study on our work. The results are discussed in Chapter 7. Finally, the conclusion can be found in Chapter 8 and the references are mentioned in the remaining pages.

1.2 Materials from Published Works

Our results for the GPT compression are published as a paper at the 60th Annual Meeting of the Association for Computational Linguistics (ACL) 2022. The title of the paper is *"Kronecker Decomposition for GPT Compression"* [Edalati et al., 2022b]. Also, our results for the PET direction are published as a paper, titled *"KronA: Parameter Efficient Tuning with Kronecker Adapter"*, on arXiv [Edalati et al., 2022a]. This thesis is written based on the two mentioned works. The thesis author, Mr. Ali Edalati is the first author of both papers.

[Edalati et al., 2022b] and [Edalati et al., 2022a] are licensed under CC BY 4.0 which allows re-using, redistribution, remix, and transformation of the materials such as figures and tables under the below conditions:

- Providing proper citation.

- Providing a link to the license.
- Mentioning if any modification is made to the original material.

All of the re-used materials (figures and tables) in this work are licensed under CC BY 4.0. To re-use these materials, we respected the license conditions. In the caption of the re-used tables or figures, their source is cited, their license link is mentioned, and the applied modifications are explained where applicable[1].

The re-used figures and tables from the published work in addition to their source and license which allows re-using are mentioned in Tables 1.1 and 1.2, respectively.

Figure	Source	License
2.7	[Devlin et al., 2019]	CC BY 4.0
2.9	[Lewis et al., 2020]	CC BY 4.0
2.10	[Lewis et al., 2020]	CC BY 4.0
2.11	[Raffel et al., 2020]	CC BY 4.0
2.14	[Chen et al., 2021]	CC BY 4.0
2.15	[Zhang et al., 2020]	CC BY 4.0
2.16	[Chen et al., 2021]	CC BY 4.0
3.2	[Tahaei et al., 2022]	CC BY 4.0

Table 1.1: This table shows the re-used figures and their corresponding source and license. Note that for each figure, re-using is allowed by its licenses.

1.3 Statement of Contribution in the Published Papers and Thesis

Prof. James J. Clark, Mehdi Rezagholizadeh, and Vahid Partovi Nia supervised the progress in addition to revising and editing [Edalati et al., 2022a,b]. Ali Edalati, Ahmad Rashid, Marzieh Tahaei, and Mehdi Rezagholizadeh contributed to the writing of [Edalati et al., 2022b]. Also, Ali Edalati, Marzieh Tahaei, and Mehdi Rezagholizadeh contributed to the

[1]In this thesis, ”GPT-2_{small}” which represents GPT-2 in the [Edalati et al., 2022b] tables is modified to ”GPT-2_{base}”. Also, the Kronecker factors are denoted by $\mathbf{A}_k$ and $\mathbf{B}_k$ in [Edalati et al., 2022a] but in this thesis, they are denoted by $\mathbf{A}_K$ and $\mathbf{B}_K$.

Table	Source	License
4.1	[Edalati et al., 2022a]	CC BY 4.0
5.1	[Edalati et al., 2022b]	CC BY 4.0
5.3	[Edalati et al., 2022a]	CC BY 4.0
5.4	[Edalati et al., 2022a]	CC BY 4.0
5.5	[Edalati et al., 2022a]	CC BY 4.0
5.6	[Edalati et al., 2022a]	CC BY 4.0
5.7	[Edalati et al., 2022a]	CC BY 4.0
5.8	[Edalati et al., 2022a]	CC BY 4.0
5.9	[Edalati et al., 2022a]	CC BY 4.0
5.10	[Edalati et al., 2022a]	CC BY 4.0
5.11	[Edalati et al., 2022a]	CC BY 4.0
5.12	[Edalati et al., 2022a]	CC BY 4.0
5.13	[Edalati et al., 2022a]	CC BY 4.0
6.2	[Edalati et al., 2022b]	CC BY 4.0
6.3	[Edalati et al., 2022b]	CC BY 4.0
6.4	[Edalati et al., 2022a]	CC BY 4.0
6.5	[Edalati et al., 2022a]	CC BY 4.0
6.6	[Edalati et al., 2022a]	CC BY 4.0
6.7	[Edalati et al., 2022a]	CC BY 4.0
7.1	[Edalati et al., 2022b]	CC BY 4.0
7.2	[Edalati et al., 2022b]	CC BY 4.0
7.3	[Edalati et al., 2022b]	CC BY 4.0
7.4	[Edalati et al., 2022b]	CC BY 4.0
7.5	[Edalati et al., 2022a]	CC BY 4.0
7.6	[Edalati et al., 2022a]	CC BY 4.0
7.7	[Edalati et al., 2022a]	CC BY 4.0

Table 1.2: This table shows the re-used tables from published works. Note that for each table, re-using is allowed by its licenses.

writing of [Edalati et al., 2022a], and Ivan Kobyzev revised it. Note that all of the co-authors have allowed the thesis author to re-use the materials from the published works in the thesis.

Marzieh Tahaei developed the code of the Kronecker layer, which is modified and used by Ali Edalati for the experiments in this work. All of the experiments in [Edalati et al., 2022b], [Edalati et al., 2022a], and the thesis are performed by Ali Edalati.

The thesis is written by Ali Edalati. Also, Ibtihel Amara helped the author translate the abstract from English to French and permitted the thesis author (Ali Edalati) to use the translated abstract in the thesis.

Chapter 2

Related Work and Background

In this chapter, Section 2.1 provides a brief background about LMs. Since Transformer-based LMs are the main focus of our work, these types of models are discussed in more details separately in Section 2.2. In Section 2.3, related works to the Model Compression direction are mentioned. Finally, Section 2.4, covers related works and baselines for the PET direction.

2.1 Language Models (LMs)

In the field of NLP, researchers have been trying to find automatic systems that can perform tasks related to a natural language. Translating from one language to another language, generating a coherent and cohesive essay about a given topic, understanding if two sentences are semantically equivalent, answering a question, and summarizing an article are all examples of NLP tasks.

To perform the NLP tasks, the automatic system should be able to either understand the lingual instances as input or generate the lingual instances as output or both. Therefore, a language model, abbreviated to LM, is used as a backbone in most of the NLP systems to help the system understand or generate the language.

Rule-based approaches were used in the early NLP systems in which a natural language was modeled with manually written rules. However, these naive approaches were extremely time-taking. Besides, they were not capable of modeling many aspects of natural language ambiguity. Meanwhile, LMs were developed as a result of data-driven approaches for providing probability indication of the word sequences [Jing and Xu, 2019, Rosenfeld, 2000].

Language models are exposed to real instances of natural language output (sentences) during the training. Generally, by increasing the size of the training corpus, the performance of LMs is improved. Since the 1980s, LMs has been utilized as the core component of the NLP systems. The main duty of an LM is to predict the probability of a sequence of words based on the knowledge that is gained by being exposed to a natural language. Statistical LMs were the first generation of data-driven models, which are developed mainly based on the Markov assumption [Bengio et al., 2003, Grave et al., 2017, Jing and Xu, 2019, Rosenfeld, 2000, Shoenfield, 1962].

Afterward, LMs that have a neural network backbone were introduced and outperformed the statistical ones. Nowadays, Transformer-based models that are considered the most recent version of neural network LMs are dominating the field. The following sections discuss the two mentioned categories of LMs.

2.1.1 Statistical Language Models

Let $s = (w_1, w_2, ..., w_N)$ be a sequence of N tokens (or words). A statistical LM computes the probability of the existence of s as a natural language output by multiplying the conditional probability of each token given its predecessors, which is extracted from the training corpus [Bahl et al., 1983, Bengio et al., 2003, Jing and Xu, 2019, Niesler and Woodland, 1996].

$$P(s) = P(w_1, w_2, ..., w_N) = P(w_1)P(w_2|w_1)...P(w_N|w_1, w_2, ..., w_{N-1})$$

Furthermore, the Markov assumption [Shoenfield, 1962] is used to simplify the problem:

$$P(w_t|w_1, ..., w_{t-1}) \approx P(w_t|w_{t-K+1}, ..., w_{t-1})$$

An LM that uses up to $K - 1$ predecessors to calculate the probability of a token is called a K-Gram LM [Katz, 1987]. Different strategies are deployed to improve the performance of the statistical LMs. However, the emergence of neural network based LMs has made statistical models obsolete.

2.1.2 Neural Network Language Models (NNLMs)

NNLMs are developed based on a neural network that predicts the next token of a given sequence of tokens [Bengio et al., 2003, Grave et al., 2017, Jing and Xu, 2019]. These models represent each token (or word) by a vector of real-valued numbers, called an embedding vector. Therefore, tokens of the input sentence that were naturally in a discrete space are projected into a continuous space which facilitates the calculation of the joint distribution function of the tokens. Furthermore, a K-Gram statistical model used for modeling a vocabulary of N tokens requires computing $N^K - 1$ parameters, which is quite challenging for large vocabularies. Instead, neural networks enable modeling large vocabularies with significantly fewer parameters. The current growing interest in NNLMs is caused by their superior performance over statistical models. Here, some types of NNLMs are mentioned [Bengio et al., 2003, Grave et al., 2017, Jing and Xu, 2019, Mikolov et al., 2010, Sundermeyer et al., 2015].

Feed Forward Language Models (FFLMs)

In this category, a feed-forward neural network is the main component of an LM. [Bengio et al., 2003] proposed a model (considered one of the first successful works) that utilizes neural networks for language modeling. Figure 2.1 shows the architecture of this model in which **C** is the weight matrix that projects the input words to the embedding space.

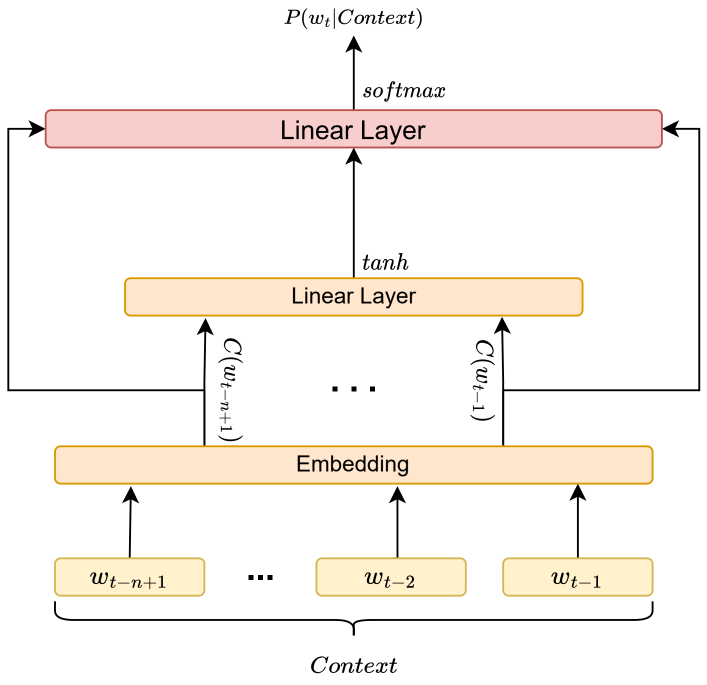


Figure 2.1: This figure shows the structure of the FFLM developed by [Bengio et al., 2003]

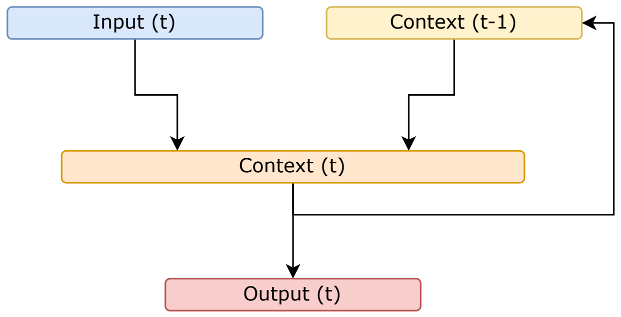


Figure 2.2: This figure depicts the workflow of the RNNLM developed by [Mikolov et al., 2010].

Recurrent Neural Network Language Models (RNNLMs)

RNNLMs utilize Recurrent Neural Networks (RNNs) [Rumelhart et al., 1986] in the structure of a LM instead of feed-forward networks [Mikolov et al., 2010, 2011a,b]. One reason is that sentences are sequential series of words (or tokens), so the order of words plays an important role in the meaning of sentences. RNNs can capture those order information better than feed-forward networks. Also, RNNs reduce the number of parameters since they use parameter sharing. In addition, the context of RNNLMs can have a flexible length which is a benefit of these models over FFLMs [Jing and Xu, 2019, Mikolov et al., 2010, Sundermeyer et al., 2015]. RNNLMs outperformed FFLMs significantly. However, RNNs fail in training on long sequences because of the gradient vanishing or explosion, which means that they cannot capture the long-range dependent information in sentences [Li et al., 2018, Sundermeyer et al., 2015].

Long-Short Term Memory (LSTM)-Based Language Models

LSTMs [Hochreiter and Schmidhuber, 1997] were designed to address the gradient vanishing (or explosion) problem of RNNs by adding a gating mechanism to the model which controls forgetting or keeping the information flow during the training. Figure 2.3 shows an LSTM module.

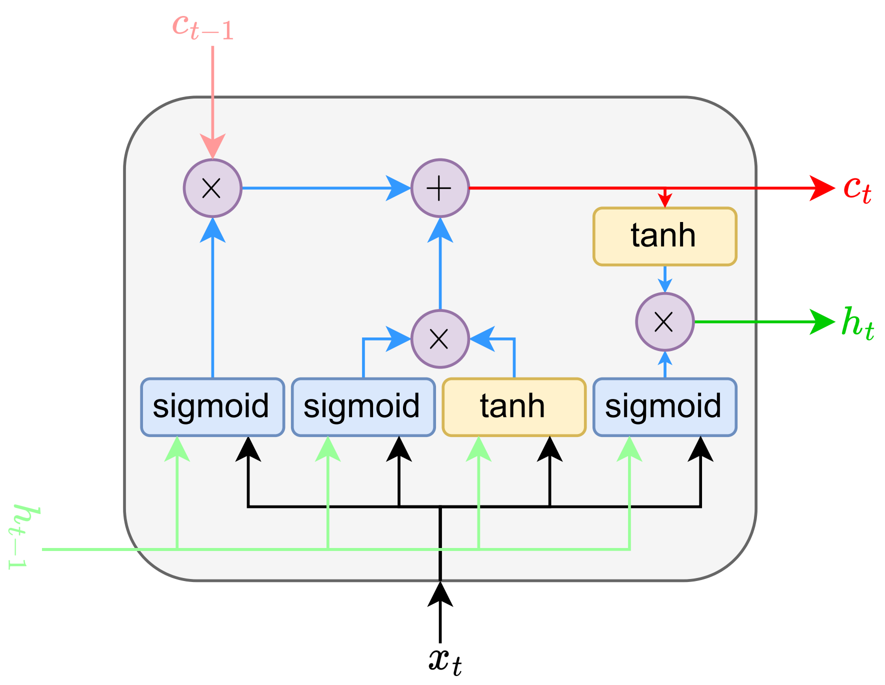


Figure 2.3: This figure depicts an LSTM [Hochreiter and Schmidhuber, 1997] module.

[Sundermeyer et al., 2012] used an LSTM as the backbone in an LM to introduce LSTM-based LMs. This model was able to capture longer dependencies than RNNLMs. However, training LSTMs is time-taking. In addition, the gradient vanishing (explosion) problem of LSTMs is not addressed completely since the training data is fed to LSTMs sequentially [Sundermeyer et al., 2015]. Finally, Transformers [Vaswani et al., 2017] were invented, which outperformed previous models in many aspects. Also, Transformers do not suffer from the gradient vanishing since the training data is fed to them in parallel, enabling them to capture long-range dependencies in the texts.

2.1.3 Masked Language Modeling (MLM) vs Casual Language Modeling (CLM)

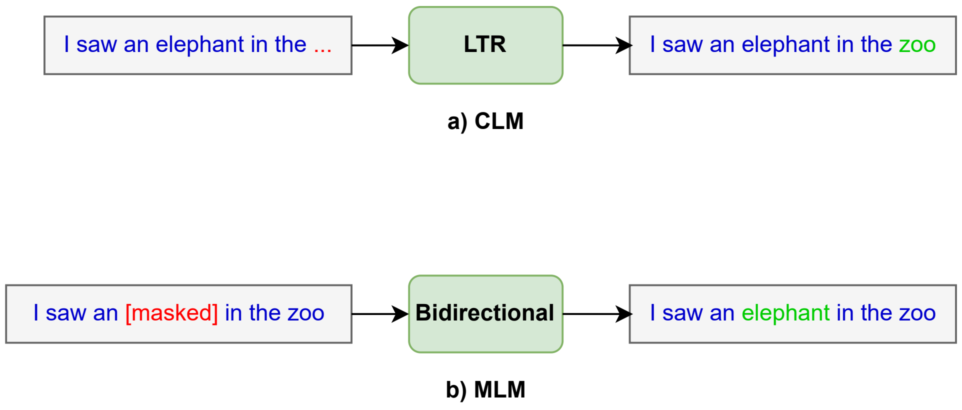

I saw an elephant in the ...
LTR
I saw an elephant in the zoo
a) CLM
I saw an [masked] in the zoo
Bidirectional
I saw an elephant in the zoo
b) MLM

Figure 2.4: This figure shows how CLM and MLM are performed

Language models can be categorized based on their method for processing the order of the input tokens. Previously, language modeling was defined as predicting the probability of a series of natural language tokens (for example, words). On the one hand, some models take a series of tokens and find the most probable token in the vocabulary to be the next token of the series. This is called CLM and a model which performs this task is called a Left To Right (LTR) model since it reads the input from left to right to predict the next token.

On the other hand, some models take the entire masked sequence of tokens (for example, a masked sentence) as the input. A masked sequence is a sequence in which some of the tokens are replaced with a masked token and the model should predict these masked tokens. This task is called MLM and a model usually should be able to read the sentence from both directions to predict the masked token. These models are categorized as Bidirectional models.

2.2 Transformer-Based Models

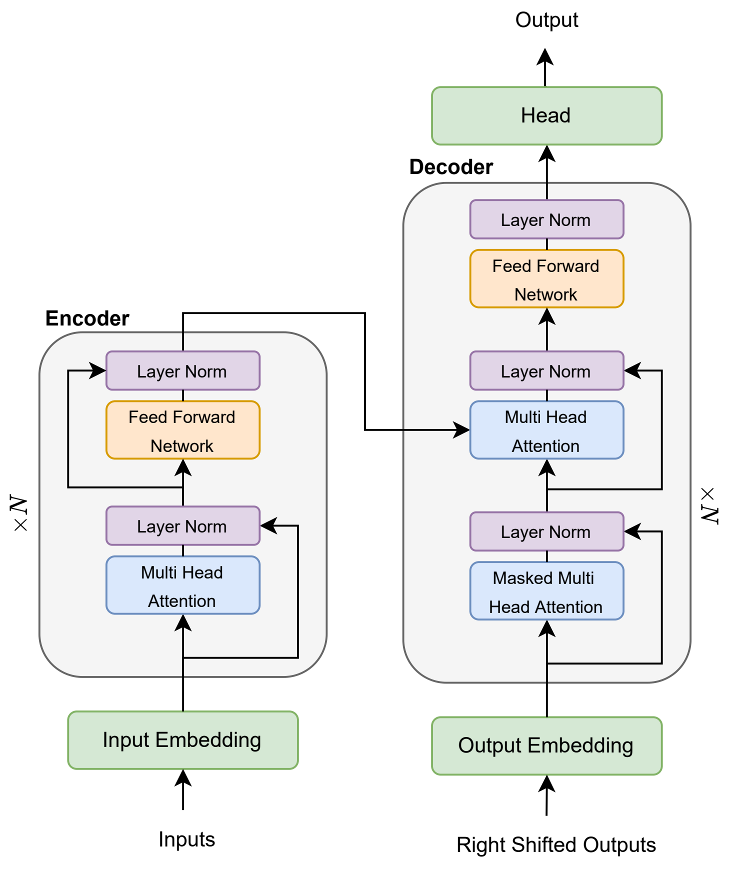


Figure 2.5: This figure shows the architecture of the Transformer-based model developed by [Vaswani et al., 2017].

[Vaswani et al., 2017] first proposed to replace RNNs and LSTMs with a novel architecture that uses the attention mechanism. The role of the attention mechanism is to find the relation and connection between all of the elements in the input sequence. For example, when a sequence of words is fed to a trained attention block, the output will be a mapping that shows for every word, how much attention is paid to all of the words in the sequence. In this mechanism, the entire input sequence is given to the model simultaneously, which addresses the problem of forgetting long-range dependencies.

This new architecture is named Transformer. Figure 2.5 illustrates the structure of the first Transformer-based model, assembled from an encoder on the left and a decoder on the right.

The goal of the encoder is to capture information about the entire input sequence. The goal of the decoder is to generate the next token of the output sequence based on the information from the encoder and the already generated tokens. Therefore, the attention block of the encoder computes the attention mappings for the entire input sentence. However, the attention block of the decoder focuses on computing the attention between already generated tokens.

Transformer layers are made from an FFN which comes after an attention block. FFN is a reversed bottleneck block which is composed of an up-projection ($\mathbf{W}_{c_{\text{fc}}} \in \mathbb{R}^{d_h \times d_{ff}}$), a non-linear activation function and a down-projection ($\mathbf{W}_{c_{\text{proj}}} \in \mathbb{R}^{d_{ff} \times d_h}$), respectively. Equation 2.1 shows how the output of the FFN is computed given the input, $\mathbf{X} \in \mathbb{R}^{l_{seq} \times d_h}$, where $\mathbf{b}_{c_{proj}} \in \mathbb{R}^{d_h}$ and $\mathbf{b}_{c_{fc}} \in \mathbb{R}^{d_{ff}}$ refer to biases in the projections, l_{seq} is the length of the input sequence, d_h is the embedding dimension of the model, and d_{ff} is the FFN's dimension. Please note that the input itself is used as a residual connection to compute the output. The residual connection is used to increase stability during the optimization process.

$$\text{FFN}(\mathbf{X}) = \text{activation}(\mathbf{X}\mathbf{W}_{c_{\text{fc}}} + \mathbf{b}_{c_{fc}})\mathbf{W}_{c_{\text{proj}}} + \mathbf{b}_{c_{proj}} + \mathbf{X} \tag{2.1}$$

In the MHA block, there are several attention heads. The output of each attention head is named $\text{head}_{i\in[1:H]}$, where H is the number of heads in the MHA. The attention head, itself is composed of linear layers which calculate the Query, Key, and Value by multiplying the input matrix, $\mathbf{X}$, by $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d_h \times d_h/H}$, respectively. Besides, there is a final linear projection, $\mathbf{W}^O \in \mathbb{R}^{d_h \times d_h}$, at the end of the block. Also, similar to the FFN, there is a residual connection in the MHA. Equations 2.2, 2.3, 2.4, 2.5, and Figure 2.6 show how the attention mechanism works in the MHA block.

$$\mathbf{Q}_i = \mathbf{X}\mathbf{W}_i^Q, \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^V, \mathbf{K}_i = \mathbf{X}\mathbf{W}_i^K \tag{2.2}$$

$$\text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{softmax}(\frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{d_h}})\mathbf{V}_i \tag{2.3}$$

$$\text{head}_i(\mathbf{X}) = \text{Attention}(\mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K, \mathbf{X}\mathbf{W}_i^V) \tag{2.4}$$

$$\text{MHA}(\mathbf{X}) = \mathbf{X} + \text{Concat}(\text{head}_1(\mathbf{X}), ..., \text{head}_n(\mathbf{X}))\mathbf{W}_i^O \tag{2.5}$$

Furthermore, the model has an embedding layer, which maps the input tokens into a set of vectors that represent the input sequence. This layer has a weight matrix (called $\mathbf{W}_{wpe} \in \mathbb{R}^{d_h \times d_h}$, d_h is the embedding/hidden dimension) which is responsible for learning the order and position of tokens in the input sequence. In addition, the embedding layer uses another weight matrix called $\mathbf{W}_{wte} \in \mathbb{R}^{v \times d_h}$ (v is the size of the vocabulary) to project each token of the input sequence to a vector. Finally, there is a head layer in Transformers that projects the output of the transformer into the desired output format. Often, the head is a linear layer plus a softmax. For more details about Transformers, see [Vaswani et al., 2017].

In the following subsections, a brief discussion of the models that are related to our experiments is provided. The source of the information related to the configuration of these models is Hugging Face[1].

[1]For more details, see https://huggingface.co/models.

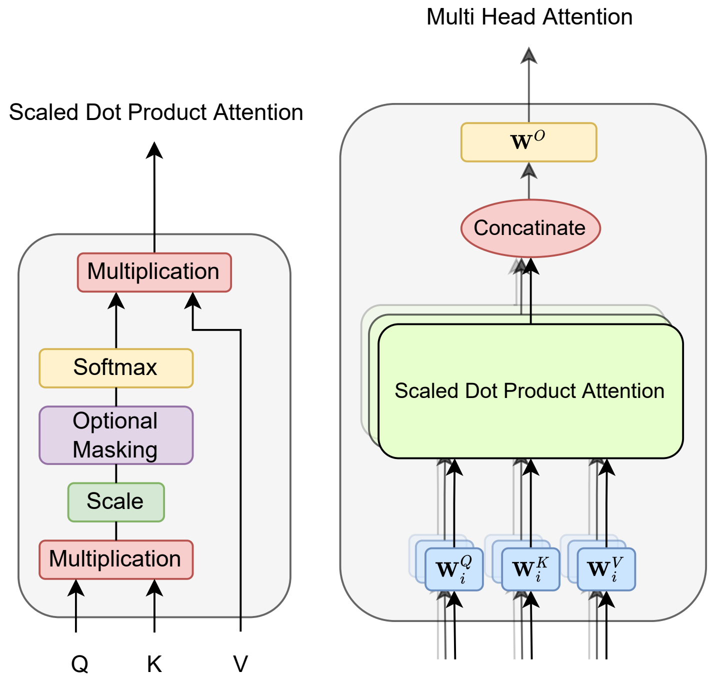


Figure 2.6: This figure shows the architecture of a MHA block on the right and the scaled dot product attention block on the left. The figure is re-drawn inspired by [Vaswani et al., 2017].

2.2.1 Bidirectional Encoder Representations from Transformers (BERT)

BERT is one the first Transformer-based LMs [Devlin et al., 2019]. This model is developed by stacking several layers of Transformer encoders in addition to an embedding layer. There are two versions for this model and Table 2.1 shows their structural configurations. Also, Figure 2.7 depicts the BERT architecture.

Name	Encoder Layers	Attention Heads	Embedding Dimension	Parameters (M)	Volume (GB)
BERT$_{\text{base}}$	12	12	768	110	0.44
BERT$_{\text{large}}$	24	16	1024	340	1.34

Table 2.1: This table shows the configuration of different versions of BERT.

This model is pre-trained on English Wikipedia and Book Corpus [Zhu et al., 2015] datasets. BERT is a bidirectional LM. Therefore, 15% of input tokens in the train data are

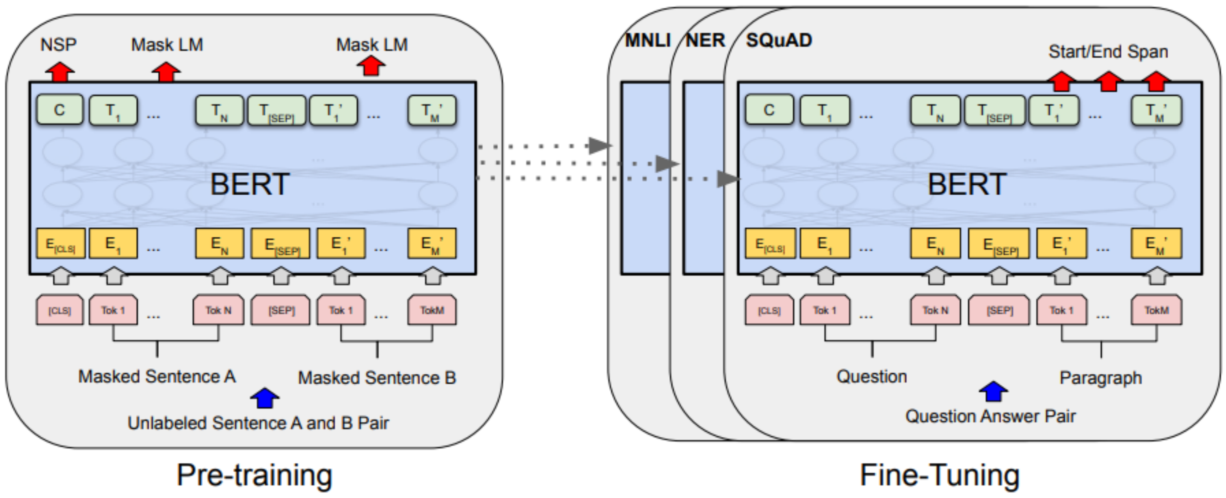


Figure 2.7: This figure shows the architecture of BERT. The figure is directly re-used with permission from [Devlin et al., 2019].

masked for pre-training and the model tries to predict them. Also, BERT is pre-trained on a classification task named Next Sentence Prediction (NSP) in which the model should determine whether the second sentence from the concatenation of two masked input sentences follows the first one. The goal of MLM is to teach the language, while the goal of NSP is to teach the relation between two consequent sentences to BERT. Please note that NSP could be beneficial for tasks like Question Answering.

Then, the model can be fine-tuned on downstream datasets like the GLUE benchmark while the gained knowledge during pre-training helps the model to achieve successful results. Please note that BERT is not a generative model so it cannot be used for those tasks that require generating a sequence of tokens as the output like Machine Translation or Summarization. Instead, it is an appropriate model for NLU tasks like the GLUE benchmark.

2.2.2 Generative Pre-trained Transformer (GPT)-2

GPT-2 [Radford et al., 2019] architecture (Figure 2.8) is composed of several identical Transformer decoders in addition to an embedding layer. GPT-2 is a generative model

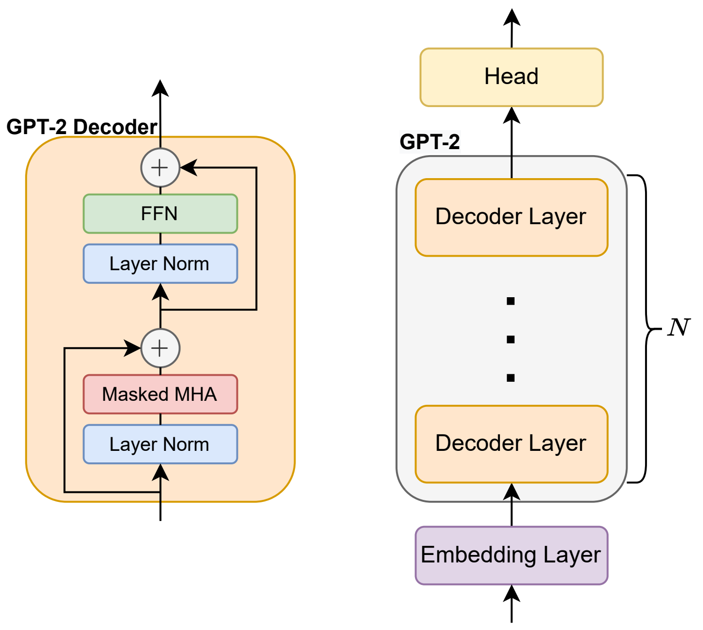


Figure 2.8: This figure shows the architecture of GPT-2

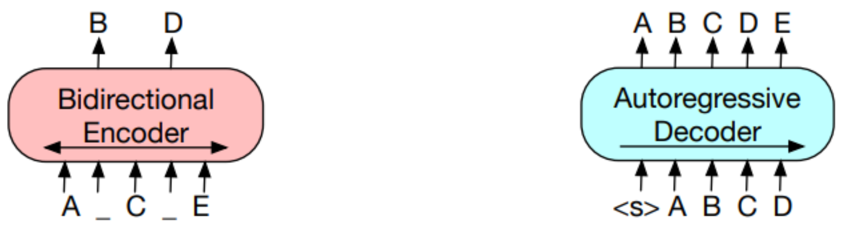
(also called auto-regressive), which means that it can be used for generation tasks like Question Answering in addition to NLU tasks. However, since it is not designed for comprehension, it usually underperforms BERT on NLU.

GPT-2 is pre-trained on a huge general domain dataset, called WebText [Radford et al., 2019], for CLM. The pre-training dataset is collected by crawling through public web pages, but it is not available to the public yet.

GPT-2 has different versions based on the number of parameters. Table 2.2 shows the structural configuration of the different GPT-2 versions. Although the proposed compression technique is only applied to GPT-2$_{\text{base}}$ in this work, it can be easily applied to the larger versions.

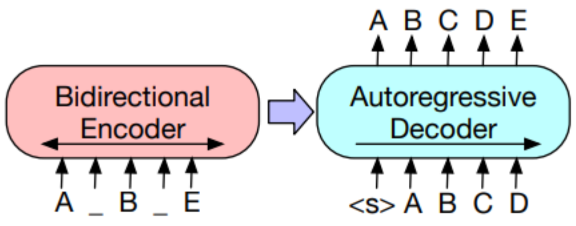
Name	Decoder Layers	Attention Heads	Embedding Dimension	Parameters (M)	Volume (GB)
GPT-2$_{\text{base}}$	12	12	768	124	0.55
GPT-2$_{\text{medium}}$	24	16	1024	355	1.52
GPT-2$_{\text{large}}$	36	20	1280	774	3.25
GPT-2$_{\text{xlarge}}$	48	25	1600	1500	6.43

Table 2.2: This table shows the configuration of the different GPT-2 versions.



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.

(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Figure 2.9: This figure shows a comparison between schematics of BERT, GPT and BART. The figure is directly re-used with permission from [Lewis et al., 2020].

2.2.3 Bidirectional and Auto-Regressive Transformer (BART)

[Lewis et al., 2020] introduced BART. This model uses a bidirectional encoder like BERT [Devlin et al., 2019], in addition to a LTR decoder like GPT [Radford et al., 2019]. Figure 2.9 shows how the output is generated using the input in BERT, GPT, and BART. Table 2.3 shows the configuration of the two BART versions that are published.

Name	Encoder Layers	Decoder Layers	Attention Heads	Embedding Dimension	Parameters (M)	Volume (GB)
$\text{BERT}_{\text{base}}$	6	6	12	768	139	0.56
$\text{BERT}_{\text{large}}$	12	12	16	1024	406	1.02

Table 2.3: This table shows the configuration of the BART versions.

For pre-training, the model should reconstruct the input data which is corrupted using the following transformations:

- **Token Masking**: Randomly selected tokens in the input are replaced with a special token called `[mask]`.

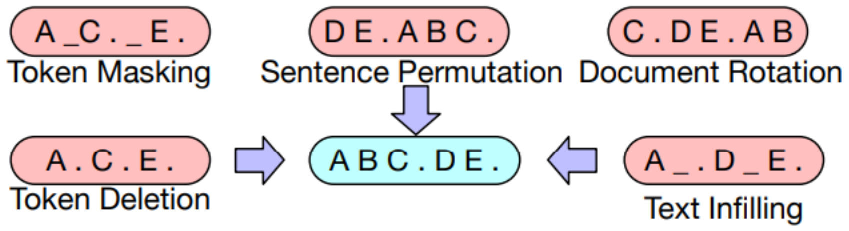


Figure 2.10: This figure shows the transformations used for pre-training data generation for the BART. The figure is directly re-used with permission from [Lewis et al., 2020].

- **Token Deletion**: Randomly selected tokens are removed from the input.
- **Text Infilling**: Inspired by [Joshi et al., 2020], randomly selected spans of tokens from the input are replaced with one `[mask]` token. The length of the span is drawn from a Poisson distribution in which $\lambda = 3$.
- **Sentence Permutation**: The order of the input sentences is shuffled randomly.
- **Document Rotation**: The input is rotated to put a randomly selected token at the beginning.

Figure 2.10 shows how the mentioned transformations corrupt the input data. The loss function is the negative log-likelihood between the intact input and the generated output.

While this model is able to perform the NLU tasks, it can perform the best on the generation tasks. Also, a multilingual version of BART - named Multi-Lingual Bidirectional and Auto-Regressive Transformer (mBART) - is developed specifically for Machine Translation by pre-training on a dataset containing sentences from different languages [Liu et al., 2020].

2.2.4 Text-to-Text Transfer Transformer (T5)

Developed by [Raffel et al., 2020], T5 is a Transformer-based model which is composed of several encoder and decoder layers similar to BART. Table 2.4 shows the configuration of the different T5 versions.

Name	Encoder Layers	Decoder Layers	Attention Heads	Embedding Dimension	Parameters (M)	Volume (GB)
$T5_{small}$	6	6	8	512	60	0.24
$T5_{base}$	12	12	12	768	220	0.89
$T5_{large}$	24	24	16	1024	770	2.95
$T5_{3B}$	24	24	32	1024	2800	11.4
$T5_{3B}$	24	24	128	1024	1100	45.2

Table 2.4: This table shows the configuration of the different T5 versions.

T5 is a generative model so it generates an output text sequence given an input text sequence. The model is pre-trained unsupervisedly on a dataset named Colossal Clean Crawled Corpus (C4) which is the cleaned version of Common Crawl[2]. Please note that the mentioned pre-training for T5 is an example of CLM.

For fine-tuning on each task, data samples should be converted to a text-to-text format. To distinguish tasks from each other, a prefix is added to the input data samples. For example, for Machine Translation from German to English, *"Translate German to English"* is added as a prefix to the input. Figure 2.11 shows how the model performs different tasks by adding an appropriate prefix to the input.

This model has different versions based on the number of layers and parameters. In our work, we used the base version with 220 Million parameters. This model represents a group of large PLMs that are difficult to fine-tune due to resource or time limitations, thus we selected this model for evaluation of the proposed PET methods. Please note that our proposed PET methods are not model dependent and they can be easily applied to any Transformer-based model.

[2]Common Crawl is a public dataset that contains web extracted texts. For more information please see here

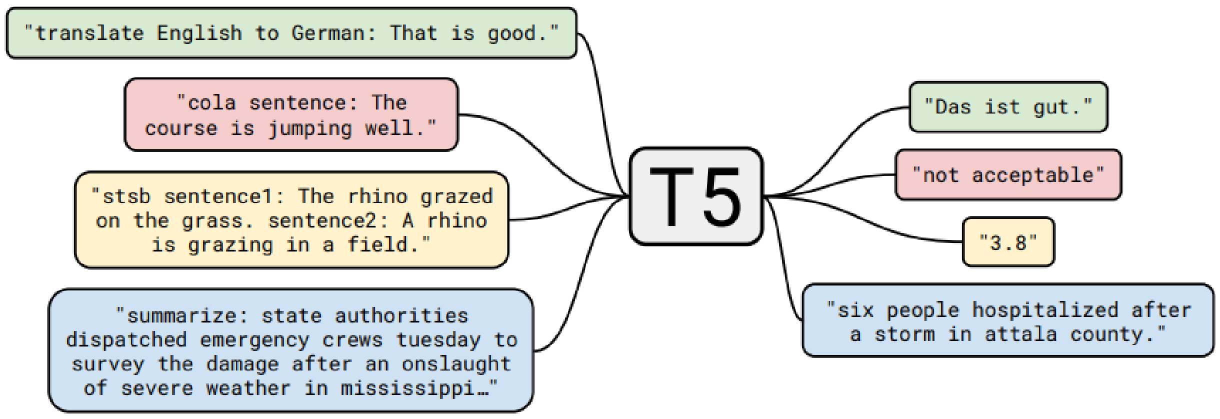

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 2.11: This figure shows how T5 converts different tasks to a text-to-text format. The figure is directly re-used with permission from [Raffel et al., 2020].

2.2.5 Generative Pre-trained Transformer (GPT)-3

Developed by OpenAI[3] [Brown et al., 2020], this model is considered one of the most recent and hugest LMs. It uses roughly the same Transformer architecture as GPT-2. However, the size of the model is considerably larger. Table 2.5 shows the configuration of the different GPT-3 versions.

Name	Number of Layers	Attention Heads	Embedding Dimension	Parameters
GPT-3$_{\text{small}}$	12	12	768	125M
GPT-3$_{\text{medium}}$	24	16	1024	350M
GPT-3$_{\text{large}}$	24	16	1536	760M
GPT-3$_{\text{xl}}$	24	24	2048	1.3B
GPT-3$_{\text{2.7B}}$	32	32	2560	2.7B
GPT-3$_{\text{6.7B}}$	32	32	4096	6.7B
GPT-3$_{\text{13B}}$	40	40	5140	13B
GPT-3$_{\text{175B}}$ or "GPT-3"	96	96	12288	175B

Table 2.5: This table shows the configuration of the different GPT-3 versions. The table is drawn based on the information provided in [Brown et al., 2020].

GPT-3 is pre-trained on the datasets that are shown in Table 2.6.

[3] https://openai.com/

Dataset	Common Crawl	WebText2	Book1	Book2	Wikipedia
Tokens Number (Billion)	410	19	12	55	3

Table 2.6: This table shows the number of tokens in the datasets used for pre-training of GPT-3. The table is drawn based on the information provided in [Brown et al., 2020].

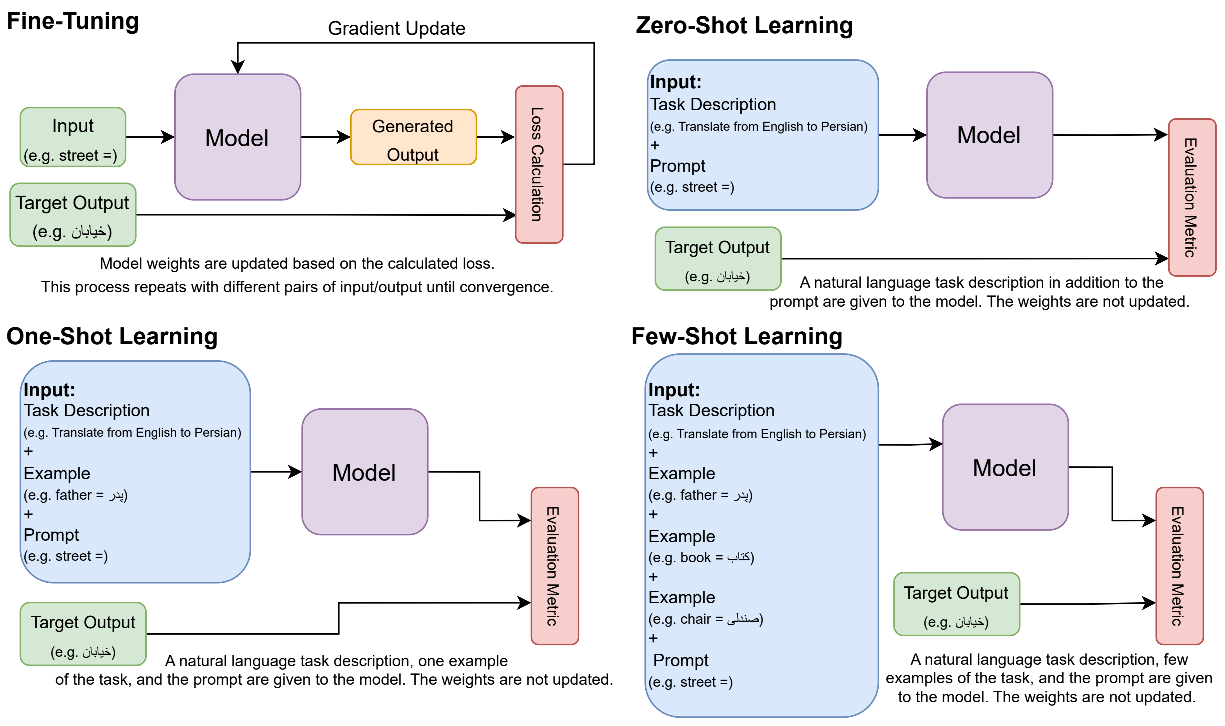

Fine-Tuning
Gradient Update
Input
(e.g. street =)
Model
Generated Output
Loss Calculation
Target Output
(e.g. خیابان)
Model weights are updated based on the calculated loss.
This process repeats with different pairs of input/output until convergence.
Zero-Shot Learning
Input:
Task Description
(e.g. Translate from English to Persian)
+
Prompt
(e.g. street =)
Model
Evaluation Metric
Target Output
(e.g. خیابان)
A natural language task description in addition to the prompt are given to the model. The weights are not updated.
One-Shot Learning
Input:
Task Description
(e.g. Translate from English to Persian)
+
Example
(e.g. father = پدر)
+
Prompt
(e.g. street =)
Model
Evaluation Metric
Target Output
(e.g. خیابان)
A natural language task description, one example of the task, and the prompt are given to the model. The weights are not updated.
Few-Shot Learning
Input:
Task Description
(e.g. Translate from English to Persian)
+
Example
(e.g. father = پدر)
+
Example
(e.g. book = کتاب)
+
Example
(e.g. chair = صندلی)
+
Prompt
(e.g. street =)
Model
Evaluation Metric
Target Output
(e.g. خیابان)
A natural language task description, few examples of the task, and the prompt are given to the model. The weights are not updated.

Figure 2.12: This diagram shows the definition of the learning based on zero-shot, one-shot, and few-shot samples compared to the fine-tuning. The diagram is drawn based on the information provided in [Brown et al., 2020].

Similar to GPT-2, this model is able to perform both NLU and generation tasks. Also, [Brown et al., 2020] showed that GPT-3 has a significantly better zero-shot, one-shot and few-shot performance compared to smaller LMs. Figure 2.12 shows the definition of zero-shot, one-shot, and few-shot learning compared to fine-tuning. The huge size of GPT-3 causes unacceptable training time and resources required for fine-tuning this model. Therefore, the few-shot learning scenario is studied for deploying GPT-3 on the downstream tasks.

2.2.6 DistilGPT2

The development process of DistilGPT2[4] is similar to DistilBERT [Sanh et al., 2019]; The parameters of DistilGPT2 are initialized randomly. Then, the model is pre-trained on OWT [Gokaslan and Cohen, 2019], which is a reproduction of the OpenAI's WebText dataset. During the pre-training, KD is used to transfer the information from the teacher - GPT-2$_{\text{base}}$ - to DistilGPT2. Thus DistilGPT2 is considered a compressed GPT-2. KD and its application as a compression technique is further discussed in Section 2.3.3. Similar to the teacher, DistilGPT2 is a LTR model which performs CLM.

It is worth mentioning that the compression of GPT-2 is not well-studied in the literature and DistilGPT2 is the only well-known compressed version of GPT-2. Therefore, it is considered the only baseline in our work. Table 2.7 shows the configuration of the DistilGPT2.

Name	Decoder Layers	Attention Heads	Embedding Dimension	Parameters (M)	Volume (GB)
DistilGPT2	6	12	768	82	0.35

Table 2.7: This table shows the configuration of DistilGPT2.

2.3 Related Works to the Model Compression Direction

In Figure 2.13, some of the well-known LMs, their number of parameters, and their release date are depicted. Just a glance is enough to conclude that the size of LMs is expanding rapidly. There are several reasons for developing such huge models, such as increasing the robustness to new domain data, improving the few-shot performance, and enabling the model to perform a wide range of tasks. Usually, when powerful computational resources are available, larger LMs are preferred since they achieve a better performance.

However, there are some scenarios in which the size of the model matters. In other words, resource limitations push using smaller models even at the expense of a slight

[4]For more details, see https://huggingface.co/distilgpt2

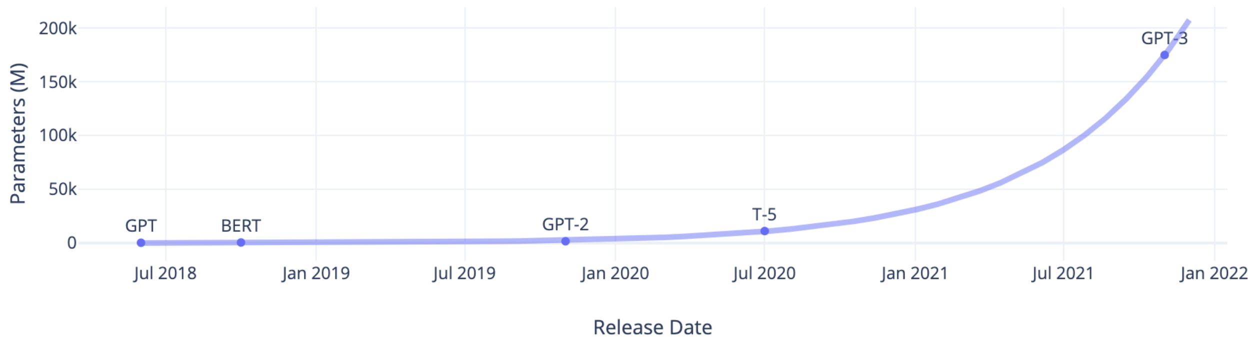


Figure 2.13: This figure shows the size of some of the well-known LMs based on their release date and number of parameters.

drop in performance. For example, enabling edge devices[5] to perform NLP tasks by using LMs is an open problem. While the performance of small LMs is not acceptable, edge devices cannot deploy huge models like GPT-3. A solution could be sending the task as a query to a central LM which is deployed on a powerful device or cloud and receiving the output. However, this solution is ineffective when the internet is not available. Also, users might be concerned about the privacy of their information. Therefore, a better solution should enable edge devices to deploy LMs with an acceptable performance.

Model Compression is a research direction that investigates different approaches to reduce the size of the models while minimizing their performance drop. In other words, Model Compression efficiently reduces the size of huge models to enable using them where resources are limited. Model Compression approaches can be divided into four categories discussed in the following subsections.

2.3.1 Pruning

Pruning compresses neural networks by removing those parameters from a model that have the most negligible effect on the final performance. Pruning methods can be divided into Structured (Channel) or Unstructured (Weight) Pruning [Liang et al., 2021]. In Structured Pruning [He et al., 2017, 2018], a group (channel) of parameters like layers or

[5]Devices that have limited computational resources, such as smartwatches and smartphones.

filters are removed while Unstructured Pruning [Gordon et al., 2020, Zhang et al., 2018] eliminates the parameters individually, resulting in a sparse network. Figure 2.14 depicts the mentioned types of Pruning.

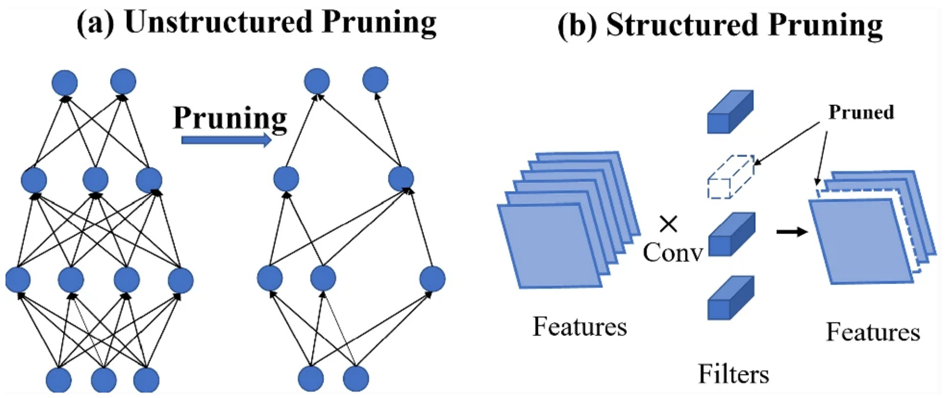


Figure 2.14: This figure shows Unstructured (a) and Structured (b) Pruning. The figure is directly re-used with permission from [Chen et al., 2021].

A simple algorithm for Pruning is mentioned here. First, a parameter or a group of parameters is temporarily removed. Second, the performance drop of the model is measured. Third, the decision about the permanent removal of the parameter(s) or preserving it (them) is made based on the desired threshold on the final performance. Finally, the mentioned steps are repeated until achieving the desired size for the compressed model [Liang et al., 2021]. In addition, further training after Pruning can be done to improve the performance [Han et al., 2016] where KD can be utilized [Chen et al., 2021] as well.

This method reduces the size of a model by removing its parameters and may result in speed-up depending on the structure of the compressed model.

2.3.2 Quantization

Normally, parameters of a neural network are in full-precision format, which means that their data type is `float32` and 32 bits are used to keep each parameter of the model. In Quantization, the data type of the parameters is changed to reduce the number of required bits for saving or loading a model, which results in speed-up and model com-

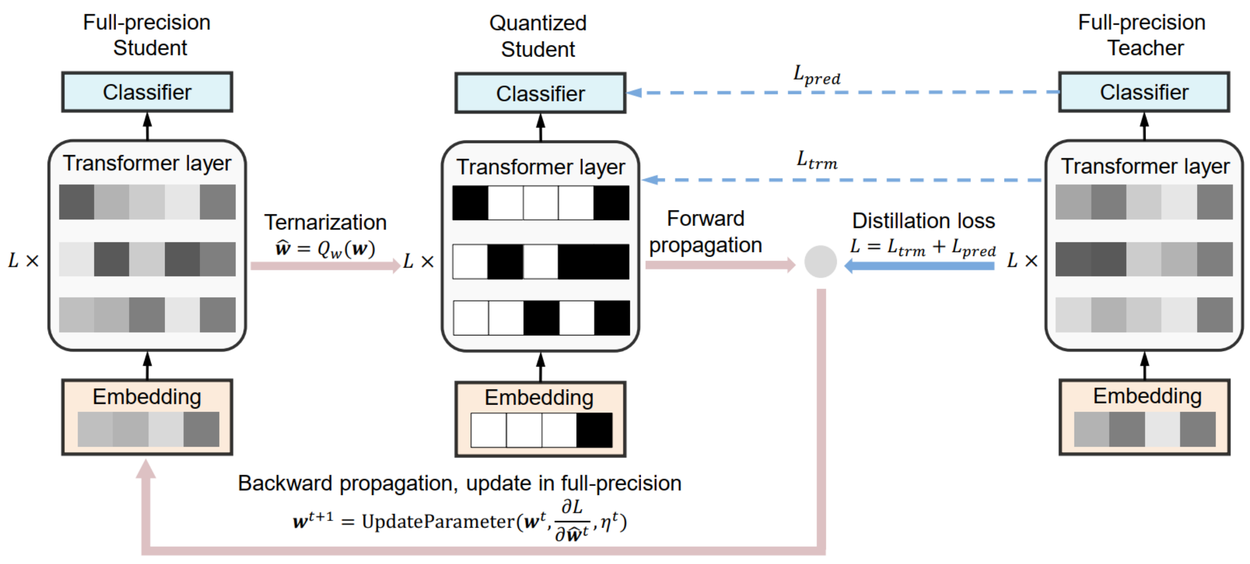


Figure 2.15: This figure shows the technique used for compressing BERT into TernaryBERT. $Q_W(\mathbf{W})$ is the operator that converts a full-precision weight matrix - $\mathbf{W}$ - into the equivalent quantized matrix, $\widehat{\mathbf{W}}$. The figure is directly re-used with permission from [Zhang et al., 2020].

pression. For example, BinaryBERT [Bai et al., 2021] and TernaryBERT [Zhang et al., 2020] are developed based on quantizing the weights of BERT into two and four bits, respectively. [Gong et al., 2014, Prato et al., 2020, Shen et al., 2020, Zafrir et al., 2019] are some works that use Quantization as a compression tool.

There are two categories of Quantization-based techniques. First, Post-Training Quantization [Liu et al., 2021] that quantizes model parameters to low bits after training. Therefore, the type of parameters during training is intact. This technique is quite fast but usually suffers from performance drop. Second, Quantization-Aware Training [Zafrir et al., 2019, Zhang et al., 2020] that trains a quantized model. While it is slower than the previous one, it can achieve better results. Also, these two Quantization methods can be combined together or with other techniques like KD to improve the performance of the compressed model [Zhang et al., 2020]. Figure 2.15 shows the compression algorithm of TernaryBERT which uses KD and Quantization.

2.3.3 Knowledge Distillation (KD)

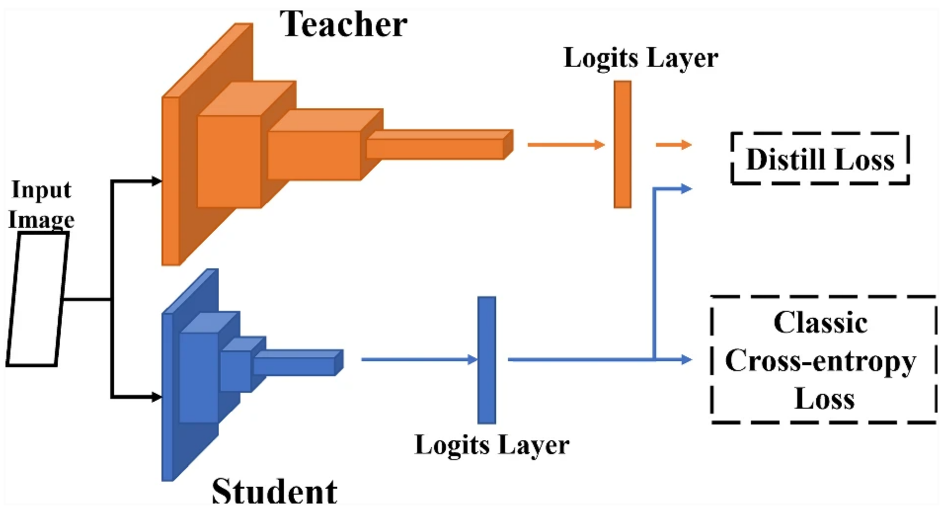


Figure 2.16: This figure shows the schematics of the vanilla response-based KD. The figure is directly re-used with permission from [Chen et al., 2021].

To train a model - named the student model - on a dataset, KD could be used as a training technique that aims to transfer information from a model that is already trained - named the teacher - to the student by forcing the student to mimic the teacher. There are three categories of KD [Gou et al., 2021]:

- **Response-based**: In this category, the distance between outputs of the student and teacher models is minimized [Hinton et al., 2015].
- **Feature-based**: This category minimizes the distance between the outputs of the last and intermediate layers - extracted features - of the student and teacher [Xu et al., 2020].
- **Relation-based**: In this category, the relationships between various pairs of layers or input samples are investigated [Yim et al., 2017].

Also, there are three schemes for the distillation [Gou et al., 2021]:

- **Offline Distillation**: The teacher is already trained and frozen. Therefore, the teacher is not updated during the distillation. [Hinton et al., 2015, Passalis and Tefas, 2018]

- **Online Distillation**: In addition to the student, the teacher is trained during the distillation. [Kim et al., 2021]

- **Self Distillation**: A single model plays the role of the teacher and student. [Huang et al., 2020, Zhang et al., 2019]

In addition, KD could be utilized as a compression technique since its goal is compressing and transferring (distillation) knowledge from a teacher, which is often the larger model, to a smaller student. To compress a model by KD, it should be used as a teacher to train a smaller student model, which is randomly initialized. However, the performance of the student is often worse than the teacher. [Jafari et al., 2021, Jiao et al., 2020, Rashid et al., 2021, Sanh et al., 2019, Sun et al., 2020] are examples of KD-based methods for compression. Also, DistilGPT2, which is discussed in Section 2.2.6, is another example of using KD for Model Compression.

2.3.4 Matrix Decomposition

Matrix Decomposition represents the weight matrices of a model by a group of smaller matrices such that the total number of parameters in the group of smaller matrices becomes less than the number of parameters in the original matrix. Therefore, the size of the model decreases.

A version of the Kronecker decomposition with summation was first used to compress convolutional and linear layers in [Wu, 2016]. Later, [Hameed et al., 2022] used another version of the Kronecker decomposition with summation to compress the convolutional layers of ResNet [He et al., 2016]. Also, [Thakker et al., 2019] applied a Kronecker-based compression technique to small LMs for Internet of Things (IoT) applications. [Thakker et al., 2020] extended the previous technique to non-IoT applications by adding some modifications.

[Tahaei et al., 2022] is one of the most recent works that applied a Kronecker-based compression technique to BERT. [Tahaei et al., 2022] can be considered a backbone in our approach. However, in chapter 6.2, we will further discuss [Tahaei et al., 2022] and the modifications we made to their approach to improve the performance and introduce ours.

To the best of our knowledge, Quantization and Pruning are not applied to compress GPT-2. Our baseline, DistilGPT2 is the result of applying KD to compress GPT-2 and this work is the first attempt to compress GPT-2 using the Kronecker decomposition.

2.4 Related Works to the PET Direction

2.4.1 Pre-training and Fine-tuning

The current dominant training paradigm of LMs on NLP tasks contains two stages:

1. **Pre-training** on a huge general domain dataset for MLM or CLM once to gain general knowledge about the targeted natural language.

2. **Fine-tuning** the pre-trained model on downstream tasks to develop a specialized model for each task.

Pre-training improves the robustness of models on out-of-domain data samples in addition to optimizing the fine-tuning stage and lowering the variance of fine-tuned models [Brown et al., 2020, Erhan et al., 2010, Hendrycks et al., 2019].

2.4.2 Fine-tuning Challenges

Previously, the growing size of LMs and its downsides on the deployment of huge models were discussed (Section 2.3). Here, another challenge related to huge models is explained: time and resource-consuming training.

By increasing the size of models, not only calculation of the forward and backward path requires more operations, but also more parameters should be updated in the training. Therefore, training huge models requires considerable time, computationally powerful devices with large memories, and notable power consumption.

The growing size of LMs makes both the pre-training and fine-tuning stages challenging. In contrast to the pre-training, which is done only once for an LM, the fine-tuning needs to be done for each downstream task. Besides, a checkpoint of the fine-tuned LM should be saved for each downstream task which requires significantly large memory since there are many downstream tasks. Therefore, a line of research - called PET - aims to make the fine-tuning efficient by lowering the number of required parameters.

The following subsections discuss a high-level overview of the related work and baselines for PET.

2.4.3 Full Fine-Tuning (FT)

This method fine-tunes the entire weights of a model on a downstream dataset. Usually, this method achieves notably good results. However, it has downsides like time and resource-consuming training. These downsides are the main motivations to develop more efficient methods.

2.4.4 BitFit Tuning

As discussed in Section 2.2, a Transformer-based model is composed of linear layers. A bias, $\mathbf{b}$, is added to the multiplication of the input vector, $\mathbf{x}$, and a weight matrix, $\mathbf{W}$, to compute the output, $\mathbf{y}$, in a linear layer (Equation 2.6). BitFit Tuning suggested tuning all or a subset of the biases while the other parameters of the model are frozen [Ben Zaken et al., 2022].

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b} \tag{2.6}$$

Although this method is relatively fast during training and does not increase the inference latency, it underperforms most of the SOTA PET techniques.

2.4.5 Prompt Tuning

Prompt Tuning suggests prepending fixed-length vectors, named prompts, to the input sequence and tuning the prompts while the original weights of a model are frozen [Lester et al., 2021, Liu et al., 2023].

Prompts can be divided into two categories of hard and soft prompts. Hard prompts are discrete tokens that usually correspond to a natural language phrase, while soft prompts are continuous vectors that may not have any meaning in the natural language. Different strategies can be followed to initialize the prompts.

This method enables storing a small number of parameters for each downstream task; thus, it is a suitable solution for switching between tasks. However, there is still more space for improving the performance of this technique to meet the SOTA.

2.4.6 Prefix Tuning

Prefix Tuning [Li and Liang, 2021] is similar to Prompt Tuning. However, it concatenates the fixed-length vectors - named prefix - to Key, $\mathbf{K}_i$, and Value, $\mathbf{V}_i$ matrices of the model [He et al., 2022a]. Also, the prefix vectors are generated by a Multi Layer Perceptron (MLP), resulting in more stable optimization and training.

Prefix Tuning outperforms Prompt Tuning. However, using the MLP adds a large number of parameters to the training phase compared to other SOTA methods. In addition, other SOTA methods can achieve better results than Prefix Tuning.

2.4.7 Adapter Tuning

[Houlsby et al., 2019] introduced Adapter Tuning. An adapter is a module inserted into a model to adapt the model for a downstream task. While the model is frozen, inserted adapters are tuned and simulate training of the model. [Houlsby et al., 2019] suggested to insert adapters sequentially after the MHA and FFN blocks.

In a basic adapter module, first, a down-projection is performed on the input matrix ($\mathbf{X} \in \mathbb{R}^{l \times d_h}$) by multiplying it by a low-rank matrix ($\mathbf{A} \in \mathbb{R}^{d_h \times r}$) and adding a bias ($\mathbf{b}_1 \in \mathbb{R}^{r}$). This reduces the dimension from d_h to r. Second, a non-linear function is applied. Third, an up-projection is applied by multiplying by another low-rank matrix ($\mathbf{B} \in \mathbb{R}^{r \times d_h}$) and adding a bias ($\mathbf{b}_2 \in \mathbb{R}^{d}_{h}$), which returns the dimension from r to d_h. Finally, the input is added as a residual connection to compute the module's output. Figure 2.17 shows where the adapters are inserted in a Transformer on the left and the structure of a basic adapter on the right. Equation 2.7 shows how the output of an adapter is computed given the input matrix, $\mathbf{X}$.

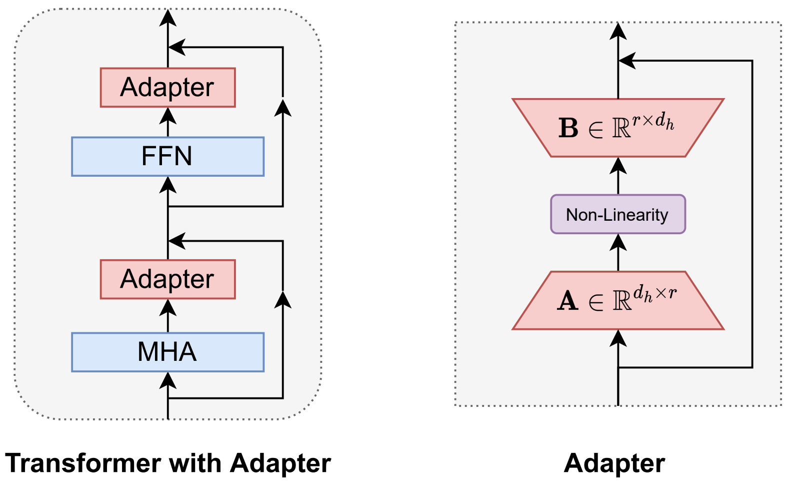

Adapter
FFN
Adapter
MHA
$\mathbf{B} \in \mathbb{R}^{r \times d_h}$
Non-Linearity
$\mathbf{A} \in \mathbb{R}^{d_h \times r}$
Transformer with Adapter
Adapter

Figure 2.17: The right figure shows the structure of a basic adapter. The left figure shows the position of the adapters that are inserted sequentially in a Transformer as suggested by [Houlsby et al., 2019]. For simplicity, the layer normalization blocks and biases are not depicted.

$$\text{Adapter}(\mathbf{X}) = \mathbf{X} + \text{NonLinearity}(\mathbf{XA} + \mathbf{b}_1)\mathbf{B} + \mathbf{b}_2 \tag{2.7}$$

This method accelerates the training while requiring a small number of trainable parameters. However, the inserted adapters increase the inference time of models and the performance is worse than the SOTA methods.

2.4.8 Compacter

Developed by [Karimi Mahabadi et al., 2021], Compacter is a modified version of the sequential adapter that is another baseline for comparison in our work. Compacter achieves acceptable performance while its design enables reducing the trainable parameters significantly. However, it requires more training and inference time than other SOTA PET methods.

In a Compacter module, the down and up projection matrices, $\mathbf{W}_{\mathrm{D}} \in \mathbb{R}^{d_h \times r}$ and $\mathbf{W}_{\mathrm{U}} \in \mathbb{R}^{r \times d_h}$, are replaced by the sum of the Kronecker product of I pairs of matrices. To further decrease the parameters, $\mathbf{A}_{\mathrm{D}_i} \in \mathbb{R}^{I \times I}$ and $\mathbf{A}_{\mathrm{U}_i} \in \mathbb{R}^{I \times I}$ matrices are shared across all of the Compacter modules. Also $\mathbf{B}_{\mathrm{D}_i} \in \mathbb{R}^{\frac{d_h}{I} \times \frac{r}{I}}$ and $\mathbf{B}_{\mathrm{U}_i}\mathbb{R}^{\frac{r}{I} \times \frac{d_h}{I}}$ matrices are generated by the multiplication of two row and column vectors (Equations 2.9 and 2.8). In addition, Compacter uses GELU as the non-linear activation function. Equation 2.10 show how Compacter works.

$$\mathbf{W}_{\mathrm{D}} = \sum_{i \in [1:I]} \mathbf{A}_{\mathrm{D}_i} \otimes \mathbf{B}_{\mathrm{D}_i} = \sum_{i \in [1:I]} \mathbf{A}_{\mathrm{D}_i} \otimes (\mathbf{s}_{\mathrm{D}_i} \mathbf{t}_{\mathrm{U}_i}^{\top}) \tag{2.8}$$

$$\mathbf{W}_{\mathrm{U}} = \sum_{i \in [1:I]} \mathbf{A}_{\mathrm{U}_i} \otimes \mathbf{B}_{\mathrm{U}_i} = \sum_{i \in [1:I]} \mathbf{A}_{\mathrm{U}_i} \otimes (\mathbf{s}_{\mathrm{U}_i} \mathbf{t}_{\mathrm{U}_i}^{\top}) \tag{2.9}$$

$$\text{Compacter}(\mathbf{X}) = \mathbf{X} + \text{GELU}(\mathbf{X}\mathbf{W}_{\mathrm{D}})\mathbf{W}_{\mathrm{U}} \tag{2.10}$$

2.4.9 Parallel Adapter (PA)

[He et al., 2022a] introduced PA, which is one of the updated versions of the basic adapters [Houlsby et al., 2019]. To the best of our knowledge, PA outperforms other versions of adapters so far; thus, PA is a baseline in our work.

To develop a PA module, four modifications are made to a basic adapter module:

- PA is inserted into the model in parallel to a PLM module while a basic adapter is inserted sequentially after a PLM module.
- Based on the empirical results, it is recommended to apply PA only to the FFN blocks.
- The output of PA is scaled by a constant factor, s, which is a hyperparameter.
- PA does not need to have a residual connection since it is inserted in parallel to FFNs, which has a residual connection.

Empirical results in [He et al., 2022a] show that these modifications significantly improve the performance and speed of PA.

Figure 2.18 shows the structure of a PA module and where it is inserted into a model. Equation 2.11 shows how the output of PA is calculated and Equation 2.12 shows how it is added to FFN to generate FFN′($\mathbf{X}$) which is the output of FFN after applying PA. [He et al., 2022a] uses ReLU as the non-linear activation function so in Figure 2.11 and Equation 2.11, ReLU is used directly.

$$\text{PA}(\mathbf{X}) = s(\max(0, \mathbf{X}\mathbf{A} + \mathbf{b}_1)\mathbf{B} + \mathbf{b}_2) \tag{2.11}$$

$$\text{FFN}'(\mathbf{X}) = \text{FFN}(\mathbf{X}) + \text{PA}(\mathbf{X}) \tag{2.12}$$

2.4.10 Low Rank Adaption (LoRA)

Similar to PA, LoRA inserts trainable modules into a PLM and only tunes them while the PLM parameters are frozen [Hu et al., 2022]. There are down and up projection matrices ($\mathbf{A} \in \mathbb{R}^{d_h \times r}$, and $\mathbf{B} \in \mathbb{R}^{r \times d_h}$) in a LoRA module. However, the non-linear activation function is removed. It also scales the output by a constant factor, s. Figure 2.19 shows

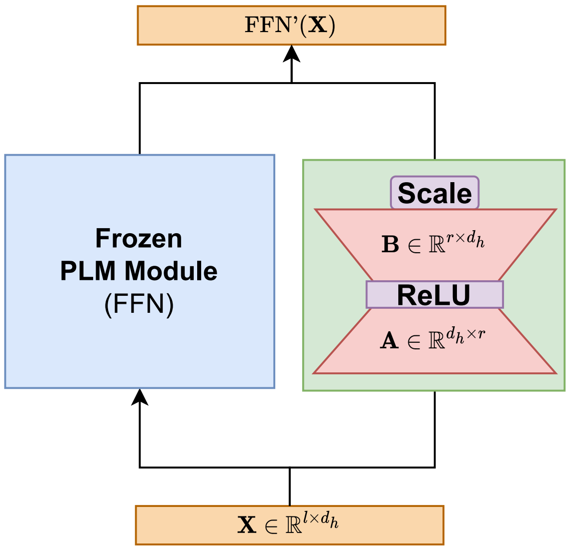

FFN'($\mathbf{X}$)
Frozen PLM Module (FFN)
Scale
$\mathbf{B} \in \mathbb{R}^{r \times d_h}$
ReLU
$\mathbf{A} \in \mathbb{R}^{d_h \times r}$
$\mathbf{X} \in \mathbb{R}^{l \times d_h}$

Figure 2.18: This figure shows the structure of a PA [He et al., 2022a] module and how it is inserted in-parallel to an FFN module.

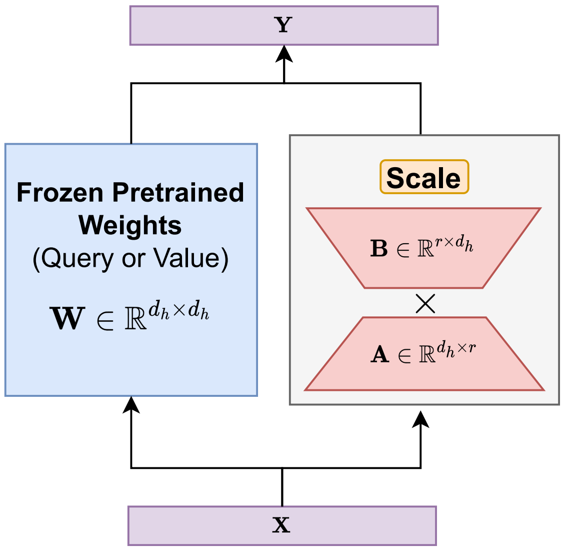

$\mathbf{Y}$
Frozen Pretrained Weights (Query or Value)
$\mathbf{W} \in \mathbb{R}^{d_h \times d_h}$
Scale
$\mathbf{B} \in \mathbb{R}^{r \times d_h}$
$\times$
$\mathbf{A} \in \mathbb{R}^{d_h \times r}$
$\mathbf{X}$

Figure 2.19: This figure depicts the structure and position of a LoRA [Hu et al., 2022] module.

the structure of the LoRA module in addition to its position in a PLM. Equation 2.13 shows how the output of LoRA is computed.

$$\text{LoRA}(\mathbf{X}) = s\mathbf{XAB} \tag{2.13}$$

There are two main differences between PA and LoRA. First, PA is inserted in parallel to FFN, which is a PLM module. Instead, LoRA can be inserted in parallel to the PLM weight matrices and [Hu et al., 2022] suggested applying them on top of the $\mathbf{W}^Q$ and $\mathbf{W}^V$ matrices in the MHA blocks. Second, due to the removal of the non-linearity and applying to matrices instead of modules, LoRA projections, $\mathbf{A}$ and $\mathbf{B}$, can be multiplied and added to the original PLM weight matrix, $\mathbf{W}$, when the training is finished. Therefore, LoRA does not add any parameters or modules to the model at the inference phase. Thus, LoRA does not increase the inference latency of the model. This is a crucial benefit of LoRA over other methods which makes it fast at the inference phase. Equation 2.14 shows how the LoRA projections are merged to generate the final weight matrix, $\mathbf{W}_{merged}$, during the inference phase.

$$\mathbf{W}_{merged} = \mathbf{W} + s\mathbf{A}\mathbf{B} \tag{2.14}$$

A disadvantage of the techniques like LoRA, PA, and Compacter is reducing the dimension, consequently the rank of the input matrix by the down-projection. Therefore the mentioned methods are rank deficient. The down-projection and rank deficiency leads to information loss, which might negatively affect the performance. We expected that this rank deficiency could be addressed by using the Kronecker product to achieve better results.

Chapter 3

Kronecker Decomposition for GPT Compression

In this chapter, our method for compressing models using Kronecker Decomposition is explained. Although we applied this method only to GPT-2, it can be simply applied to any model that has linear layers.

3.1 Kronecker Product and Decomposition

The Kronecker product is a mathematical operation that results in a block matrix given two input matrices (Equation 3.1). Let $\mathbf{A} \in R^{a_1 \times a_2}$ and $\mathbf{B} \in R^{b_1 \times b_2}$ be the inputs. The Kronecker product of $\mathbf{A}$ and $\mathbf{B}$ is equal to $\mathbf{W} \in R^{w_1 \times w_2}$. The block (i, j) of $\mathbf{W}$ is equal to $a_{i,j}\mathbf{B}$. Therefore, the shape of $\mathbf{W}$ $(w_1 \times w_2)$ is equal to the multiplication of the shapes of the input matrices where $w_1 = a_1 \times b_1$ and $w_2 = a_2 \times b_2$ [Edalati et al., 2022a,b].

$$\mathbf{W} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B}, \end{bmatrix} \tag{3.1}$$

Therefore, a large matrix can be represented as the Kronecker product of two smaller matrices, which leads to parameter reduction. In our work, this idea is used for Model Compression.

Also, the Kronecker product has the following features:

$$\mathbf{A} \otimes (\mathbf{B} + \mathbf{C}) = \mathbf{A} \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{C}$$

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$$

$$(\mathbf{A} \otimes \mathbf{B})^{\top} = \mathbf{A}^{\top} \otimes \mathbf{B}^{\top}$$

for more details see [Henderson et al., 1983].

Furthermore, we utilized another feature of the Kronecker product to reduce the number of Floating Point Operations Per Second (FLOPS) in the PET direction. Equation 3.2 shows this feature which enables calculating the output of the Kronecker adapters without the direct reconstruction of $(\mathbf{A} \otimes \mathbf{B})$. This reduces the required FLOPS based on Equation 3.3 and speeds up the calculation. In Equation 3.2, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{x}$ are the Kronecker factors and input vector, respectively. Also, $\gamma(.)$ is an operator that stacks the columns of the input matrix to reshape it into a vector. In contrast, $\eta_{m \times n}(.)$ reshapes the input vector $(e.g., \mathbf{x} \in \mathbb{R}^{mn})$ into a matrix (e.g., $\mathbf{X} \in \mathbb{R}^{m \times n}$)

$$(\mathbf{A} \otimes \mathbf{B})\mathbf{x} = \gamma(\mathbf{B}\eta_{b_2 \times a_2}(\mathbf{x})\mathbf{A}^{\top}) \tag{3.2}$$

$$\frac{(2a_1b_1 - 1)a_2b_2}{\min\big((2b_2 - 1)b_1a_2 + (2a_2 - 1)b_1a_1, (2a_2 - 1)b_2a_1 + (2b_2 - 1)b_1a_1\big)} \tag{3.3}$$

Besides, one feature of the Kronecker product motivated us to apply it in the PET direction. In all of the previously discussed adapters, the rank of the input matrix is decreased since it is multiplied by a low-rank down projection. This reduces the representation power of the adapters and makes them rank deficient. However, the rank reduction does not happen if the Kronecker product and Equation 3.2 are used in adapters, instead. Therefore, we expected to improve the performance of the existing adapters by incorporating the Kronecker product.

3.2 Linear Layer Compression Using Kronecker Decomposition

Equation 3.4 shows how the output of a linear layer, $\mathbf{y} \in \mathbb{R}^{w_1}$, is calculated given the input, $\mathbf{x} \in \mathbb{R}^{w_2}$, where $\mathbf{b} \in \mathbb{R}^{w_1}$ is a bias and $\mathbf{W} \in \mathbf{R}^{w_1 \times w_2}$ is a weight matrix. To compress this layer, the weight matrix can be represented by the Kronecker product of two smaller matrices (called Kronecker factors), $\mathbf{A} \in \mathbf{R}^{a_1 \times a_2}$ and $\mathbf{B} \in \mathbf{R}^{b_1 \times b_2}$, such that $\mathbf{W} = \mathbf{A} \otimes \mathbf{B}$ and $w_1 = a_1 b_1$, $w_2 = a_2 b_2$. Therefore, the number of parameters is decreased from $w_1 w_2$ to $a_1 a_2 + b_1 b_2$. For example, let $w_1 = w_2 = 1024$, $a_1 = a_2 = 2$, and $b_1 = b_2 = 512$. Then, the compression rate will be computed based on Equation 3.5.

$$\mathbf{W}\mathbf{x} + \mathbf{b} = \mathbf{y} \tag{3.4}$$

$$\frac{1024^2}{512^2 + 2^2} \approx 4 \tag{3.5}$$

If we represent the weight matrix by two Kronecker factors using Kronecker Decomposition, Equation 3.4 changes to Equation 3.6.

$$(\mathbf{A} \otimes \mathbf{B})\mathbf{x} + \mathbf{b} = \mathbf{y} \tag{3.6}$$

We can simply reconstruct $\mathbf{W}$ by obtaining $\mathbf{A} \otimes \mathbf{B}$ and multiply it by $\mathbf{x}$ to compute the output of the Kronecker layer. It is possible to use Equation 3.2 to further accelerate Equation 3.6. However, using this technique requires GPUs with a larger memory during training. Since our resources were limited, for GPT compression we did not use Equation 3.2. Instead, we simply reconstructed $\mathbf{A} \otimes \mathbf{B}$ directly.

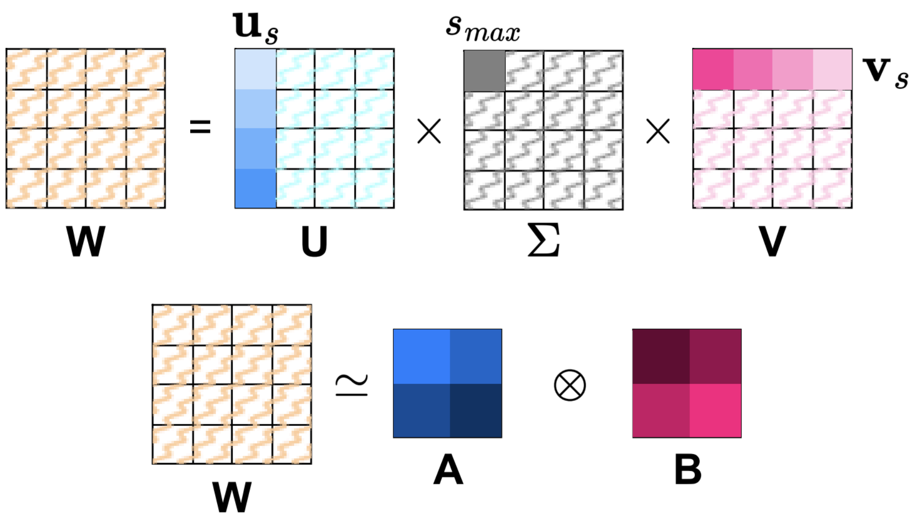


Figure 3.1: This figure shows how SVD is used to decompose a matrix - $\mathbf{W}$ - into the Kronecker factors such that $\mathbf{W} \simeq \mathbf{A} \otimes \mathbf{B}$.

To estimate the corresponding Kronecker factors for a given $\mathbf{W}$, we used a pair of Kronecker factors $(\mathbf{A}, \mathbf{B})$ that minimize the L2 loss function stated in Equation 3.7. To solve Equation 3.7, SVD of $\mathbf{W}$ was calculated (Equation 3.8) where $\mathbf{U}$ is the left singular vectors matrix, $\boldsymbol{\Sigma}$ is the singular values matrix and $\mathbf{V}$ is the right singular vectors matrix. S_{max} is the largest singular value. Also, $\mathbf{u}_s$ and $\mathbf{v}_s$ are the corresponding column and row to S_{max} in $\mathbf{U}$ and $\mathbf{V}$ matrices, respectively. $\mathbf{u}_s$ and $\mathbf{v}_s$ were multiplied by $\sqrt{s_{max}}$, then reshaped as $(a_1 \times a_2)$ and $(b_1 \times b_2)$, respectively, to be exploited as the corresponding Kronecker factors (Equations 3.9 and 3.10). Figure 3.1 shows the described process for decomposing $\mathbf{W}$ into the Kronecker factors. Obviously, this decomposition method is not completely precise and $\mathbf{W} \simeq \mathbf{A} \otimes \mathbf{B}$. Please see [Tahaei et al., 2022, Van Loan, 2000] for more details.

$$(\mathbf{A}, \mathbf{B}) = \underset{(\hat{\mathbf{A}}, \hat{\mathbf{B}})}{\arg\min} \|\mathbf{W} - \hat{\mathbf{A}} \otimes \hat{\mathbf{B}}\|^2 \tag{3.7}$$

$$\mathbf{W} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V} \tag{3.8}$$

$$\mathbf{A} = \text{reshape}_{(a_1 \times a_2)}(\sqrt{S_{max}}\mathbf{u}_s) \tag{3.9}$$

$$\mathbf{B} = \text{reshape}_{(b_1 \times b_2)}(\sqrt{S_{max}}\mathbf{v}_s) \tag{3.10}$$

3.3 Embedding Layer Compression Using Kronecker Decomposition

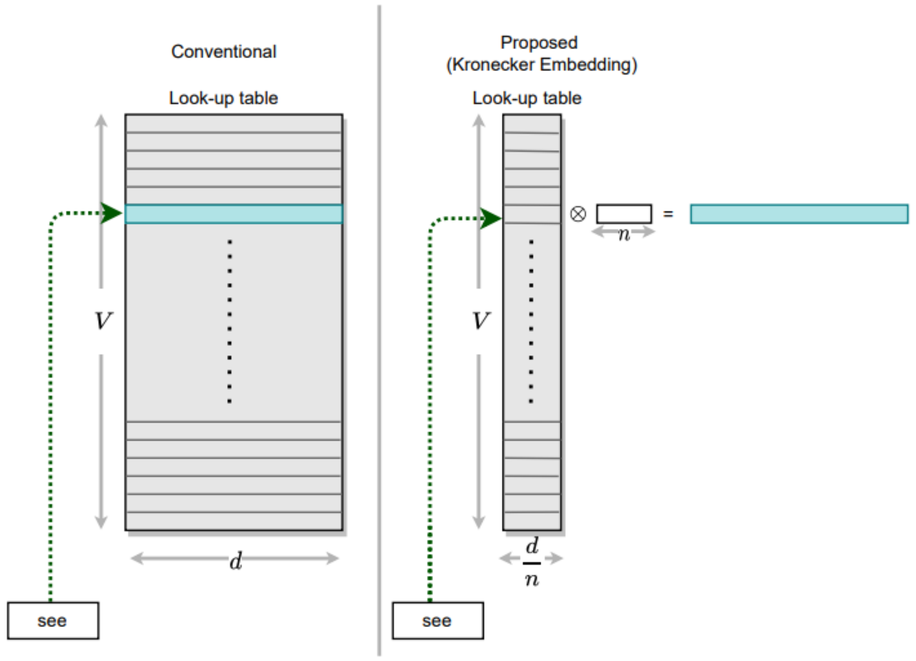


Figure 3.2: This figure illustrates our method for the embedding layer compression. The figure is directly re-used with permission from [Tahaei et al., 2022].

In the modern LMs, the size of $\mathbf{W}_{wte} \in \mathbb{R}^{v \times d_h}$ is significantly larger than $\mathbf{W}_{wpe} \in \mathbb{R}^{d_h \times d_h}$ since $v \gg d_h$. Therefore, $\mathbf{W}_{wpe}$ was compressed similarly to the linear layers while the compression of $\mathbf{W}_{wte}$ was slightly different.

The corresponding Kronecker factors to $\mathbf{W}_{wte}$ are $\mathbf{A}_{wte} \in \mathbb{R}^{v \times d/n}$ and $\mathbf{b}_{wte} \in \mathbb{R}^{1 \times n}$. We represented the second Kronecker factor with a vector instead of a matrix for two reasons. First, to set the shape of $\mathbf{A}_{wte}$ similar to $\mathbf{W}_{wte}$ so that each token is embedded using a single row in $\mathbf{A}_{wte}$. Second, $\mathbf{A}_{wte}(i,:) \otimes \mathbf{b}_{wte}$ is equal to the embedding vector of the i^{th} token. The computation complexity of $\mathbf{A}_{wte}(i,:) \otimes \mathbf{b}_{wte}$ becomes efficient ($\mathcal{O}(d)$). This algorithm is introduced by [Tahaei et al., 2022]. Figure 3.2 shows how the embedding compression works.

3.4 GPT2 Compression Using Kronecker Decomposition

Section 2.2 explains how Transformers are developed based on linear layers. Also, Sections 3.2 and 3.3 discuss how Kronecker Decomposition is applied to compress linear and embedding layers. We used Kronecker decomposition to compress the embedding and linear layers of GPT-2, which is a Transformer-based model. We call the new compressed model KnGPT2. Figure 3.3 shows how our method decomposes GPT-2 into KnGPT2.

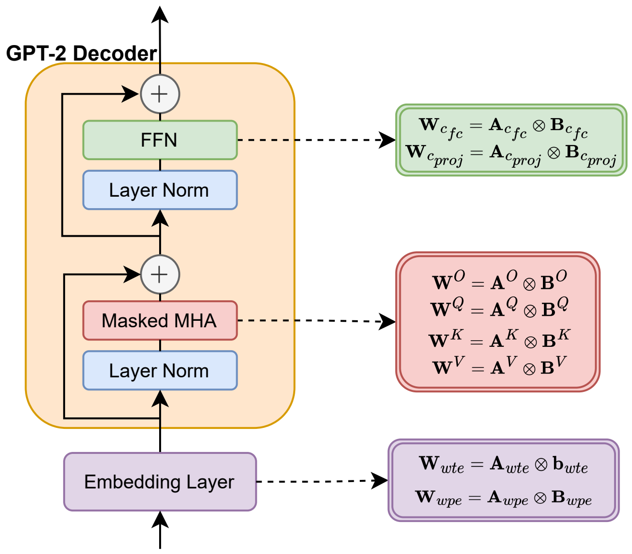

GPT-2 Decoder
FFN
Layer Norm
Masked MHA
Layer Norm
Embedding Layer
$\mathbf{W}_{c_{fc}} = \mathbf{A}_{c_{fc}} \otimes \mathbf{B}_{c_{fc}}$
$\mathbf{W}_{c_{proj}} = \mathbf{A}_{c_{proj}} \otimes \mathbf{B}_{c_{proj}}$
$\mathbf{W}^O = \mathbf{A}^O \otimes \mathbf{B}^O$
$\mathbf{W}^Q = \mathbf{A}^Q \otimes \mathbf{B}^Q$
$\mathbf{W}^K = \mathbf{A}^K \otimes \mathbf{B}^K$
$\mathbf{W}^V = \mathbf{A}^V \otimes \mathbf{B}^V$
$\mathbf{W}_{wte} = \mathbf{A}_{wte} \otimes \mathbf{b}_{wte}$
$\mathbf{W}_{wpe} = \mathbf{A}_{wpe} \otimes \mathbf{B}_{wpe}$

Figure 3.3: This figure shows how different linear layers of GPT-2 are decomposed into the Kronecker factors to generate the compressed model, KnGPT2.

3.5 Intermediate Layer Knowledge Distillation (ILKD)

As discussed in Section 3.2, Kronecker Decomposition is not accurate. Therefore, the performance of the decomposed GPT-2 is notably worse than the original model. However, the lost performance can be retrieved by a brief pre-training.

Inspired by [Jiao et al., 2020, Tahaei et al., 2022], we developed an ILKD technique which is utilized at the pre-training and fine-tuning stages. Our ablation study (Section 6.1.2) shows that using the ILKD is essential to achieve the best results.

Term	Definition
S	The student model, KnGPT2
T	The teacher model, GPT-2
$(\mathbf{X}, \mathbf{Y})$	A batch of data
$\text{Att}^{S}_{last}(\mathbf{X})$	The output of the student's last attention block
$\text{Att}^{T}_{last}(\mathbf{X})$	The output of the teacher's last attention block
$\text{H}^{S}_{l}(\mathbf{X})$	The normalized output of the student's l^{th} layer
$\text{H}^{T}_{l}(\mathbf{X})$	The normalized output of the teacher's l^{th} layer
$\mathcal{L}_{\text{CE}}(\mathbf{X}, \mathbf{Y})$	Cross Entropy (CE) between the model's output, given $\mathbf{X}$ as the input, and $\mathbf{Y}$
$\text{K-L}\{\mathbf{X}\|\|\mathbf{X}'\}$	Kullback–Leibler (K-L) divergence between $\mathbf{X}$ and $\mathbf{X}'$
$\text{MSE}(\mathbf{X}, \mathbf{X}')$	Mean Squared Error (MSE) between $\mathbf{X}$ and $\mathbf{X}'$
$\alpha_1, \alpha_2, \alpha_3$	The scales used in the loss function
L	Number of the hidden states

Table 3.1: This table defines the terms used in our ILKD algorithm.

Please see Table 3.1 for the definition of the terms used in this section. Inspired by [Wang et al., 2020], K-L divergence between the outputs of the models' last attention block (Equation 3.11) was used in our algorithm. Also, we utilized MSE between the hidden states of the teacher and student (Equation 3.12) to match the student's intermediate outputs to the teacher's. The output of the embedding layer or each Transformer layer was considered a hidden state. In addition, CE between the ground truth outputs and generated outputs was calculated.

As Equation 3.13 shows, the final loss function is obtained by adding the three weighted loss items.

$$\mathcal{L}_{\text{Attention}}(\mathbf{X}) = \text{K-L}\{\text{Att}^{S}_{last}(\mathbf{X})||\text{Att}^{T}_{last}(\mathbf{X})\} \tag{3.11}$$

$$\mathcal{L}_{\text{Hidden States}}(\mathbf{X}) = \frac{1}{L}\sum_{l} \text{MSE}\{\text{H}^{S}_{l}(\mathbf{X}), \text{H}^{T}_{l}(\mathbf{X})\} \tag{3.12}$$

$$\mathcal{L}_{\text{total}}(\mathbf{X},\mathbf{Y}) = \sum_{(\mathbf{X},\mathbf{Y})} \alpha_1 \mathcal{L}_{\text{Attention}}(\mathbf{X}) + \alpha_2 \mathcal{L}_{\text{Hidden States}}(\mathbf{X}) + \alpha_3 \mathcal{L}_{\text{CE}}(\mathbf{X},\mathbf{Y}) \tag{3.13}$$

Chapter 4

Parameter Efficient Tuning with Kronecker Adapter

This chapter explains how we used the Kronecker product to develop modules that are used in efficient tuning of PLMs.

4.1 Kronecker Adapter (KronA)

The structure of a LoRA module is depicted in Figure 4.1 where d_h is the embedding dimension, r is the rank of LoRA that is selected based on the desired number of training parameters, $\mathbf{A} \in \mathbb{R}^{d_h \times r}$ is the down-projection matrix, and $\mathbf{B} \in \mathbb{R}^{r \times d_h}$ is the up-projection matrix. Equation 4.1 shows how the output of LoRA is calculated given the input, $\mathbf{X} \in \mathbb{R}^{l_{seq} \times d_h}$, where s is the scale.

$$\text{LoRA}(\mathbf{X}) = s\mathbf{XAB} \tag{4.1}$$

A KronA module is developed by replacing the LoRA projections with the Kronecker factors, $\mathbf{A}_K \in \mathbb{R}^{a_1 \times a_2}$ and $\mathbf{B}_K \in \mathbb{R}^{b_1 \times b_2}$, which are randomly initialized. Instead of matrix multiplication, the Kronecker factors are multiplied by the input using the Kronecker product (Equation 3.2). Based on the desired number of trainable parameters, the shapes of the Kronecker factors are selected as a hyperparameter. Table 4.1 summarizes the men-

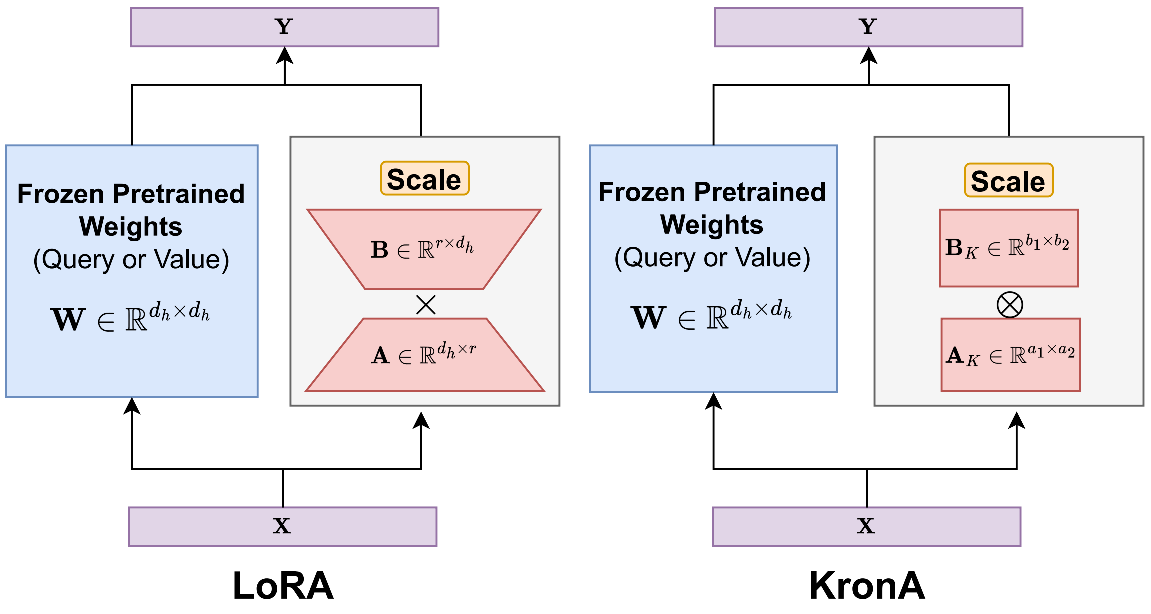

Y
Frozen Pretrained Weights (Query or Value)
$\mathbf{W} \in \mathbb{R}^{d_h \times d_h}$
Scale
$\mathbf{B} \in \mathbb{R}^{r \times d_h}$
$\times$
$\mathbf{A} \in \mathbb{R}^{d_h \times r}$
X
LoRA
Y
Frozen Pretrained Weights (Query or Value)
$\mathbf{W} \in \mathbb{R}^{d_h \times d_h}$
Scale
$\mathbf{B}_K \in \mathbb{R}^{b_1 \times b_2}$
$\otimes$
$\mathbf{A}_K \in \mathbb{R}^{a_1 \times a_2}$
X
KronA

Figure 4.1: This figure shows the structure of LoRA [Hu et al., 2022] (left) and KronA [Edalati et al., 2022a] (right) module.

tioned information. Equation 4.2 shows how the output of a KronA module is computed, where s is the scale.

$$\text{KronA}(\mathbf{X}) = s\mathbf{X}[\mathbf{A}_K \otimes \mathbf{B}_K] \tag{4.2}$$

Module Name	Factor Name	Symbol	Shape	Parameters	Module Parameters	Constraint
KronA	Kronecker Factor	$\mathbf{A}_K$	$a_1 \times a_2$	$a_1 a_2$	$a_1 a_2 + b_1 b_2$	$a_1 b_1 = a_2 b_2 = d_h$
	Kronecker Factor	$\mathbf{B}_K$	$b_1 \times b_2$	$b_1 b_2$		
LoRA	Down Projection	$\mathbf{A}$	$d_h \times r$	$d_h r$	$2 d_h r$	$r < \frac{d_h}{2}$
	Up Projection	$\mathbf{B}$	$r \times d_h$	$d_h r$		

Table 4.1: This table compares the features of LoRA with KronA. The table is re-used with permission from [Edalati et al., 2022a].

KronA is a linear block that is inserted in parallel to the linear weight matrices of a PLM. Therefore, similar to LoRA, KronA modules can be merged into the PLM after training. Therefore, the inference latency is not increased by using KronA. Equation 4.3 shows the merging mechanism.

$$\mathbf{W}_{merged} = \mathbf{W} + s[\mathbf{A}_K \otimes \mathbf{B}_K] \tag{4.3}$$

Therefore, both LoRA and KronA do not increase the inference time.

For the initialization, we set $\mathbf{B}_K$ to zero and initialize $\mathbf{A}_K$ from a Kaming-Uniform (KU) distribution [He et al., 2015]. In Section 6.2.1, a brief ablation study on the effect of the weight initialization is done.

4.2 Kronecker Adapter for Blocks (KronA$^{\mathrm{B}}$)

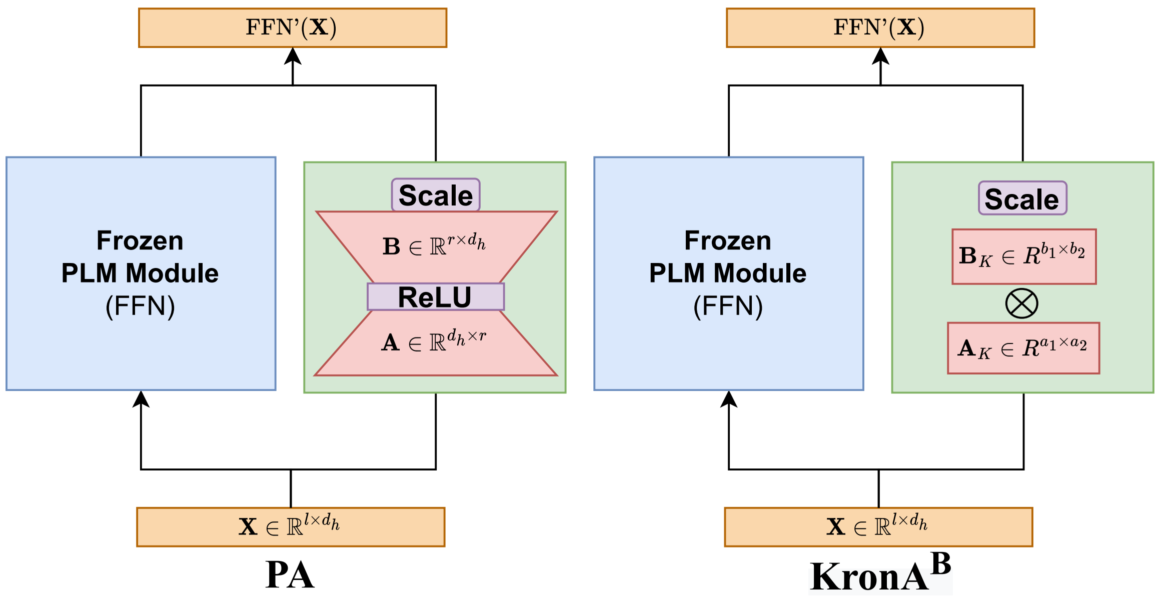

FFN'(X)
Frozen PLM Module (FFN)
Scale
$\mathbf{B} \in \mathbb{R}^{r\times d_h}$
ReLU
$\mathbf{A} \in \mathbb{R}^{d_h\times r}$
$\mathbf{X} \in \mathbb{R}^{l\times d_h}$
PA
FFN'(X)
Frozen PLM Module (FFN)
Scale
$\mathbf{B}_K \in R^{b_1\times b_2}$
$\otimes$
$\mathbf{A}_K \in R^{a_1\times a_2}$
$\mathbf{X} \in \mathbb{R}^{l\times d_h}$
KronA$^{\mathrm{B}}$

Figure 4.2: This figure depicts the structure of a PA module [He et al., 2022a] on the left and a KronA$^{\mathrm{B}}$ [Edalati et al., 2022a] module on the right.

[He et al., 2022a] proposed PA which also uses a low-rank decomposition in-parallel to FFNs and achieves a promising performance on downstream applications. Equations 4.4 and 4.5 show how PA works in a Transformer-based model. Also, the structure of a PA module is depicted in Figure 4.2.

$$\mathrm{PA}(\mathbf{X}) = s(\max(0, \mathbf{XA} + \mathbf{b_1})\mathbf{B} + \mathbf{b_2}) \tag{4.4}$$

$$\text{FFN}'(\mathbf{X}) = \text{FFN}(\mathbf{X}) + \text{PA}(\mathbf{X}) \tag{4.5}$$

The promising performance of PA motivated us to develop a Kronecker-based version of PA. Therefore, we replaced the PA projections with the Kronecker factors. Also, our empirical results show that removing the non-linearity from KronA$^{\text{B}}$ improves the performance and speed. Therefore, the non-linearity is removed from KronA$^{\text{B}}$ blocks (see Section 6.2.2 for more details). Equations 4.6 and 4.7 show how the output of KronA$^{\text{B}}$ is computed and added to the FFN's output to generate the final output, FFN′$(\mathbf{X})$, where $\mathbf{b}$ is a bias.

$$\text{KronA}^{\text{B}}(\mathbf{X}) = s\mathbf{X}[\mathbf{A}_K \otimes \mathbf{B}_K] + \mathbf{b} \tag{4.6}$$

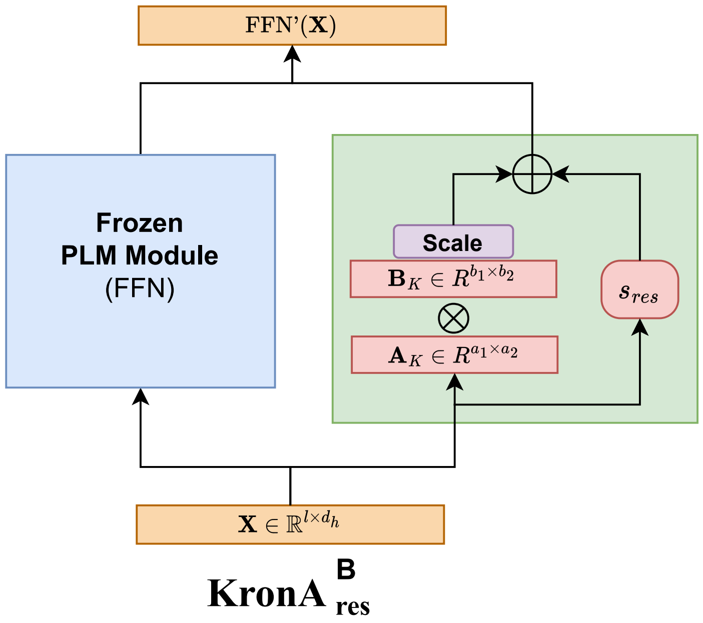
$$\text{FFN}'(\mathbf{X}) = \text{FFN}(\mathbf{X}) + \text{KronA}^{\text{B}}(\mathbf{X}) \tag{4.7}$$

Although KronA$^{\text{B}}$ is a linear block, there is a non-linear activation function within FFNs which prevents merging the KronA$^{\text{B}}$ blocks into FFNs during the inference phase. Therefore, similar to PA, KronA$^{\text{B}}$ increases the inference latency and memory.

4.3 Kronecker Adapter for Blocks with Residual Connection (KronA$^{\text{B}}_{\text{res}}$)

We developed a third version of our Kronecker adapter, which is slower than the previous versions but achieves a better performance. This adapter is developed by adding a scaled residual connection to the KronA$^{\text{B}}$ module. This residual connection was scaled by a learnable factor, s_{res}, which is initialized with one at the beginning of fine-tuning. Figure 4.3 shows the structure of a KronA$^{\text{B}}_{\text{res}}$ module. Also, Equation 4.8 shows how the output of KronA$^{\text{B}}_{\text{res}}$ is calculated.

$$\text{KronA}^{\text{B}}_{\text{res}}(\mathbf{X}) = s\mathbf{X}[\mathbf{A}_K \otimes \mathbf{B}_K] + \mathbf{b} + s_{res}\mathbf{X} \tag{4.8}$$



FFN'(**X**)
Frozen
PLM Module
(FFN)
Scale
$\mathbf{B}_K \in R^{b_1 \times b_2}$
$\otimes$
$\mathbf{A}_K \in R^{a_1 \times a_2}$
s_{res}
$\mathbf{X} \in \mathbb{R}^{l \times d_h}$
KronA $^{\text{B}}_{\text{res}}$

Figure 4.3: This figure shows the structure of a KronA$^{\text{B}}_{\text{res}}$ [Edalati et al., 2022a] module.

Chapter 5

Experimental Details

In this chapter, first, the datasets used in our experiments are explained in Section 5.1. Then, the evaluation metrics are discussed in Section 5.2. Finally, Sections 5.3 and 5.4 are dedicated to explaining the experimental details (including hyperparameters, hardware, etc.) about the Model Compression and PET directions, respectively.

5.1 Datasets

5.1.1 Open Web Text (OWT)

WebText [Radford et al., 2019] is a huge general domain dataset that is collected by OpenAI[1]. The original version of this dataset is not published yet. However, a replication of this dataset called OWT [Gokaslan and Cohen, 2019] has been published[2]. This dataset contains eight million documents in the training set. [Li et al., 2021] proposed an algorithm to clean the dataset from noisy data including the HTML codes or sentences with a high ratio of non-alphabetical characters. We used the cleaned version of OWT. Also, only the first 10% of data samples in the dataset were selected as our pre-training dataset in this work.

[1]See https://openai.com/
[2]See Hugging Face and OpenWebTextCorpus.

5.1.2 WikiText-103

WikiText-103 [Merity et al., 2017] is a language modeling dataset with more than 100 million tokens. Containing numbers, punctuation, a relatively large vocabulary, and the original case of words are considered among the advantages of this dataset. We used WikiText-103 to evaluate the models on CLM.

5.1.3 General Language Understanding Evaluation (GLUE)

GLUE is a well-known benchmark in NLP that includes ten downstream tasks [Wang et al., 2019]. In our work, two (WNLI and AX) out of ten GLUE tasks were excluded since they have too small datasets leading to unstable and inconclusive results. Here, a short description of the utilized tasks is provided [Wang et al., 2019]:

- **Corpus of Linguistic Acceptability (CoLA)**: This is an English dataset collected from articles on linguistic theory. The model should predict if a sample sentence is grammatically correct [Warstadt et al., 2019].

- **Multi-Genre Natural Language Inferenc (MNLI)**: An English dataset that includes data samples from different sources. Each data sample has a hypothesis and a premise sentence. The model should predict whether the hypothesis is entailed by the premise [Williams et al., 2018].

- **Microsoft Research Paraphrase Corpus (MRPC)**: An English dataset extracted from the online news in which every data sample is composed of two sentences. The task is predicting if the two sentences are equivalent in terms of semantics [Dolan and Brockett, 2005].

- **Question-Answering Natural Language Inference (QNLI)**: A dataset developed based on SQuAD [Rajpurkar et al., 2016] that has changed the Question-Answering task into a classification task. Data samples in QNLI are composed of an article and a question and the task predicting if the answer to the question can be found in the article [Demszky et al., 2018].

- **Quora Question Pairs (QQP)**: A dataset collected from question pairs found in Quora[3]. The task is about predicting if the two input questions are equivalent in terms of their semantics.

- **Recognizing Textual Entailment (RTE)**: This dataset is collected from the news and Wikipedia. Data samples contain a hypothesis and a premise sentence and the model should predict whether the hypothesis is entailed by the premise [Bar Haim et al., 2006, Bentivogli et al., 2009, Dagan et al., 2006, Giampiccolo et al., 2007].

- **Stanford Sentiment Treebank-2 (SST-2)**: This dataset is made from comments about movies and the task is to predict if the sentiment of the comment is positive or negative [Socher et al., 2013].

- **Semantic Textual Similarity Benchmark (STS-B)**: This dataset is developed by extracting sentence pairs from the news headlines and captions of images and videos. The task is to assign a number from 1 to 5 to show the similarity of the sentence pairs [Agirre et al., 2007].

The GLUE tasks have three splits: train, dev, and test. In our Model Compression direction, all models were trained on the train set, then, evaluated on the dev set to tune the hyperparameters. Finally, they were tested on the test set by submitting the results to the GLUE leaderboard[4]. Please note that the original test labels of GLUE are not published so the test evaluation requires submitting the predictions to the GLUE leaderboard, which is time-taking. Therefore, for the PET experiments, we generated our test sets from the dev and train data similar to [Karimi Mahabadi et al., 2021, Zhang et al., 2022]. To generate the test set of tasks with a relatively larger dataset (MNLI, QNLI, QQP, SST-2), we

[3]See https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs
[4]See https://gluebenchmark.com/

excluded 1k randomly selected samples from the train set. for other tasks (CoLA, RTE, MRPC, STS-B), half of the dev set were considered as the test set.

5.2 Evaluation Metrics

The metrics that are utilized for evaluation are discussed in this section.

5.2.1 Perplexity (PPL)

PPL is a metric to measure the ability of LMs to generate semantically and syntactically correct sentences. By generating more natural and correct sentences, PPL of an LM decreases. Equation 5.1 shows the formula to measure PPL of an LM where $X = (x_0, x_1, ..., x_N)$ is a sequence generated by the model and $p_\theta(x_n|X_{n-1})$ means the probability of generating x_n when the model has already generated $X_{n-1} = (x_0, x_1, ..., x_{n-1})$ [Bahl et al., 1983, Clarkson and Robinson, 1999]. A lower PPL means that the model generates the sentences that are found in the test set (assumed to be semantically and syntactically correct) by a higher probability. This metric is used for the evaluation on WikiText-103.

$$\text{PPL}(X) = e^{-\frac{1}{N}\sum_{n=0}^{N} \log p_\theta(x_n|X_{n-1})} \tag{5.1}$$

5.2.2 Accuracy (Acc)

This metric is usually used for the evaluation of classifier models. Acc [Powers, 2011] simply reports the percentage of those data samples that are classified correctly divided by the total data samples (Equation 5.2). This metric is used for the evaluation on RTE, MNLI, QNLI, QQP and SST-2.

$$\text{Acc} = \frac{\text{Number of samples classified correctly}}{\text{Number of total samples}} \times 100 \tag{5.2}$$

5.2.3 F1

F1 is often used to evaluate the classification ability of models. In our work, F1 was used to evaluate the models on MRPC. To calculate F1 for a classifier's output, the below terms must be defined first [Powers, 2011]:

- **True Positive (TP)** is the number of data samples assigned to a class correctly.
- **True Negative (TN)** is the number of data samples not assigned to a class correctly.
- **False Positive (FP)** is the number of data samples assigned to a class incorrectly.
- **False Negative (FN)** is the number of data samples not assigned to a class incorrectly.

Now, Equations 5.3 and 5.4 show how two other related metrics - named Precision and Recall - for each class are defined.

$$\text{Precision}(class) = \frac{\text{TP}(class)}{\text{TP}(class) + \text{FP}(class)} \tag{5.3}$$

$$\text{Recall}(class) = \frac{\text{TP}(class)}{\text{TP}(class) + \text{FN}(class)} \tag{5.4}$$

Finally, Equation 5.5 shows how F1 for each class is defined based on Recall and Precision. Note that in our work, the F1 percentage was reported, so it was scaled by 100.

$$\text{F1}(class) = \frac{2 \times \text{Precision}(class) \times \text{Recall}(class)}{\text{Precision}(class) + \text{Recall}(class)} \tag{5.5}$$

5.2.4 Matthews's Correlation Coefficient (MCC)

MCC [Chicco et al., 2021, Matthews, 1975] is a metric that measures the similarity of the predicted labels to the true labels. it is commonly used to evaluate classifiers on CoLA.

Equation 5.6 shows how MCC is calculated.

$$\text{MCC} = \frac{\text{TN} \times \text{TP} - \text{FN} \times \text{FP}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{5.6}$$

5.2.5 Pearson's Correlation Coefficient (PCC)

The goal of STS-B is to measure the similarity of two sentences by assigning a number. Therefore, it is a regression task that requires an appropriate metric. PCC [Rodgers and Nicewander, 1988] is a metric that can measure the similarity of two series of numbers. Let $\mathbf{y} = (y_1, ...y_N)$ and $\mathbf{x} = (x_1, ...x_N)$ be the true and predicted numbers for the similarity of N sentences, respectively. Then, Equation 5.7 shows how this metric is calculated in which:

$$\mu_{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\mu_{\mathbf{y}} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

.

$$\text{PCC}(\mathbf{y}, \mathbf{x}) = \frac{\sum_{i=1}^{N}(x_i - \mu_{\mathbf{x}})(y_i - \mu_{\mathbf{y}})}{\sqrt{\sum_{i=1}^{N}(x_i - \mu_{\mathbf{x}})^2}\sqrt{\sum_{i=1}^{N}(y_i - \mu_{\mathbf{y}})^2}} \tag{5.7}$$

5.2.6 Spearman's Rank Correlation Coefficient (SCC)

Similar to PCC, SCC [Dodge, 2008, Spearman, 1904] is a metric to measure the similarity of two series of numbers. Assume that $\mathbf{x} = (x_1, ...x_N)$ is a series of numbers and $\mathbf{R}(\mathbf{x}, i)$ is an operator which sorts the vector $\mathbf{x}$ and returns the order of the element x_i in the sorted vector. Therefore, SCC of two series of numbers, $\mathbf{x}$ and $\mathbf{y}$, is defined as Equation 5.8 shows.

$$\text{SCC}(\mathbf{x}, \mathbf{y}) = 1 - \frac{6\sum_{i=1}^{N}(\mathbf{R}(\mathbf{x}, i) - \mathbf{R}(\mathbf{y}, i))^2}{N(N^2 - 1)} \tag{5.8}$$

We reported the average of PCC and SCC to evaluate the models on STS-B.

5.3 Experimental Details Related to the Model Compression Direction

All of the experiments were done on NVIDIA Tesla V100 GPUs, using Pytorch[5] [Paszke et al., 2019] and the Huggingface Transformers library [Wolf et al., 2020]. While we used eight GPUs for experiments related to the pre-training or fine-tuning on WikiText-103, one GPU is used for each GLUE experiment.

Our baseline, DistilGPT2 has roughly 82 million parameters. We wanted to compress the base model, GPT-2 with 124 million parameters, to the size of the baseline for a fair comparison. Therefore, odd-numbered Transformer layers besides the embedding layer of GPT-2 were decomposed by a factor of 2. In other words, only half of the Transformer layers and the embedding layer were compressed to initialize KnGPT2. In this scenario, KnGPT2 has 83 million parameters.

Table 5.1 shows the sizes of the matrices in GPT-2$_{\text{base}}$, DistilGPT2 and KnGPT2. The hyperparameters that we used for Model Compression experiments are provided in Table 5.2.

Model	Embedding	Q,K,V	FFN[6]
GPT-2$_{\text{base}}$	50527×768	768×768	3072×768
DistilGPT2	50527×768	768×768	3072×768
KnGPT2[7]	$\mathbf{A} : 50527 \times 384, \mathbf{B} : 1 \times 2$	$\mathbf{A} : 384 \times 768, \mathbf{B} : 2 \times 1$	$\mathbf{A} : 1536 \times 768, \mathbf{B} : 2 \times 1$

Table 5.1: The shape of the weight matrices of the models studied in the Model Compression direction are mentioned in this table. The table is directly re-used with permission from [Edalati et al., 2022b].

[5]See https://github.com/pytorch/pytorch

[6]Due to the space limitation, only the shape of $\mathbf{W}_{c_{\text{proj}}}$ is mentioned. The shape of $\mathbf{W}_{c_{\text{fc}}}$ is in the reversed order of $\mathbf{W}_{c_{\text{proj}}}$.

[7]Only the shape of decomposed weight matrices is mentioned. Half of the Transformer layers were not decomposed.

Phase	Dataset	Model	Epoch	Seq Length	Batch size	Learning rate	α_1	α_2	α_3	α_4
P	OWT(10%)	KnGPT2	1	1024	1	0.00025	0.5	0.5	0.1	0
F	WikiText-103	KnGPT2	1	1024	1	0.00025	0.5	0.5	0.1	0
F	WikiText-103	DistilGPT2	1	1024	1	0.00025	0	0	1	0
F	Wikitext-103	GPT-2	1	1024	1	0.00025	0	0	1	0
F	GLUE	GPT-2	20	128	16	2e-5	0	0	1	0
F	GLUE	DistilGPT2	20	128	16	2e-5	0	0	1	0
F	GLUE	DistilGPT2+KD	20	128	16	2e-5	0	0	0.5	0.5
F	GLUE	KnGPT2	20	128	16	2e-5	0	0	1	0
F	GLUE	KnGPT2+ILKD	20	128	16	2e-5	0.5	0.5	0.02	0

Table 5.2: This table shows the hyperparameters used for the pre-training (shown by P) and the fine-tuning (shown by F) of the models used in the Model Compression direction. Also, α_4 represents the scale of the vanilla KD loss for fine-tuning DistilGPT2 with KD.

5.4 Experimental Details Related to the PET Direction

5.4.1 Experimental Setup

All of the discussed PET methods were applied to fine-tune T5 [Raffel et al., 2020]. Our code implementation was based on Hugging Face Transformers[8] [Wolf et al., 2020], LoRA[9] [Hu et al., 2022], PA[10] [He et al., 2022a], and Compacter[11] [Karimi Mahabadi et al., 2021] official public repositories. Similar to the mentioned repositories, our code was developed based on Pytorch. One NVIDIA Tesla V100 GPU was utilized to run the PET experiments.

Except for BitFit Tuning, the other techniques used approximately the same number of trainable parameters. Therefore, we can have a fair comparison. Note that it was not possible to increase the trainable parameter of BitFit tuning due to the limited number of biases in T5. Furthermore, KronA$^{\mathrm{B}}$ or KronA$^{\mathrm{B}}_{\mathrm{res}}$ modules can benefit from one or two bias vectors, which can slightly change the number of trainable parameters. For each GLUE task, the final performance indicated whether to use one or two biases.

Also, Table 5.3 shows the position of the studied PET modules in the model.

[8]See https://github.com/huggingface/transformers
[9]See https://github.com/microsoft/LoRA
[10]See https://github.com/jxhe/unify-parameter-efficient-tuning
[11]See https://github.com/rabeehk/compacter

Method	FT	BitFit	Adapter	Compacter	LoRA	PA	KronA	KronA^B	KronA$^B_{res}$
Block	Entire Model	Biases	FFN & Attention	FFN & Attention	Query & Value	FFN	Query & Value	FFN	FFN

Table 5.3: This table shows the position of the PET modules in a Transformer-based model. The table is directly re-used with permission from [Edalati et al., 2022a].

5.4.2 Hyperparameters

Based on the target number of trainable parameters, the Kronecker factors of our proposed adapters can have various shapes. The shapes of the Kronecker factors of KronA^B and KronA$^B_{res}$ were arbitrarily chosen due to the time limitation. However, we tuned the shape of the Kronecker factors, reported in KronA experiments, based on the best dev results. The options that were investigated as the shape of Kronecker factors and their resulting accuracy on MNLI are reported in Table 5.4 to show the effect of the Kronecker factors shape on the final performance. In this example, $(2, 384)$ and $(384, 2)$ are selected as the shape of $\mathbf{A}_K$ and $\mathbf{B}_K$, respectively.

Shape of $\mathbf{A}_K$	Shape of $\mathbf{B}_K$	MNLI (Accuracy)
(48, 16)	(16, 48)	86.50
(32, 24)	(24, 32)	86.31
(3, 256)	(256, 3)	86.16
(24, 32)	(32, 24)	86.40
(2, 384)	(384, 2)	**86.63**
(192, 4)	(4, 192)	86.46
(12, 64)	(64, 12)	86.56

Table 5.4: The investigated options for the shape of KronA Kronecker factors and their corresponding MNLI accuracy is reported in this table. The table is adapted with permission from [Edalati et al., 2022a]. The column "Shape of $\mathbf{B}_K$" is added to this table compared to the original table.

The hyperparameters that are used for each experiment are provided in Tables 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, and 5.13. To reproduce the reported performances of FT, BitFit, and Compacter in [Karimi Mahabadi et al., 2021], we used the exact set of hyperparameters reported in [Karimi Mahabadi et al., 2021]. For LoRA, PA, and Adapter experiments, the rank was selected differently from [Karimi Mahabadi et al., 2021] to match the trainable parameters with other baselines. We also tuned the learning rate for the mentioned in addition to our proposed methods. The checkpoint that achieved the best dev results during 20 epochs of training was evaluated on the test set.

FT hyperparameters					
Task	learning rate	batch size	warmup steps	source sentence length	epoch
GLUE	3e-4	100	500	128	20

Table 5.5: The hyperparameters of FT experiments are reported in this table. The table is directly re-used with permission from [Edalati et al., 2022a].

BitFit hyperparameters					
Task	learning rate	batch size	warmup steps	source sentence length	epoch
GLUE	3e-4	100	500	128	20

Table 5.6: The hyperparameters of BitFit Tuning experiments are reported in this table. The table is directly re-used with permission from [Edalati et al., 2022a].

Adapter hyperparameters				
Task	learning rate	batch size	task reduction factor	epoch
GLUE	3e-3	100	32	20

Table 5.7: The hyperparameters of Adapter experiments are reported in this table. The table is directly re-used with permission from [Edalati et al., 2022a].

Compacter hyperparameters					
Task	learning rate	batch size	hypercomplex division	task reduction factor	epoch
GLUE	3e-3	100	4	32	20

Table 5.8: The hyperparameters of Compacter experiments are reported in this table. The table is directly re-used with permission from [Edalati et al., 2022a].

PA hyperparameters					
Task	learning rate	batch size	rank	scale	epoch
CoLA	3e-3	100	2	16	20
RTE	5e-3	100	2	16	20
MRPC	5e-3	100	2	16	20
SST-2	1e-3	100	2	16	20
STS-B	1e-3	100	2	16	20
MNLI	1e-3	100	2	16	20
QNLI	1e-3	100	2	16	20
QQP	1e-3	100	2	16	20

Table 5.9: The hyperparameters of PA experiments are reported in this table. The table is re-used with permission from [Edalati et al., 2022a]. In the original table, the name of STS-B is written SSTS-B, which is corrected here.

LoRA hyperparameters					
Task	learning rate	batch size	rank	s	epoch
GLUE	1e-3	100	1	1	20

Table 5.10: The hyperparameters of LoRA experiments are reported in this table. The table is directly re-used with permission from [Edalati et al., 2022a].

KronA hyperparameters						
Task	learning rate	batch size	$\mathbf{A}_K$	$\mathbf{B}_K$	s	epoch
CoLA	1e-3	100	(32,24)	(24,32)	1	20
RTE	2e-3	100	(32,24)	(24,32)	1	20
MRPC	1e-3	100	(32,24)	(24,32)	1	20
SST-2	1e-3	100	(24,32)	(32,24)	1	20
STS-B	1e-3	100	(2,384)	(384,2)	1	20
MNLI	1e-3	100	(2,384)	(384,2)	1	20
QNLI	1e-3	100	(3,256)	(256,3)	1	20
QQP	1e-3	100	(24,32)	(32,24)	1	20

Table 5.11: The hyperparameters of KronA experiments are reported in this table. The table is re-used with permission from [Edalati et al., 2022a]. In the original table, the name of STS-B is written SSTS-B, which is corrected here.

KronA^B hyperparameters							
Task	learning rate	batch size	$\mathbf{A}_K$	$\mathbf{B}_K$	s	module bias	epoch
CoLA	1e-3	100	(32,24)	(24,32)	16	2	20
RTE	5e-3	100	(32,24)	(24,32)	16	1	20
MRPC	5e-3	100	(32,24)	(24,32)	16	1	20
SST-2	1e-3	100	(32,24)	(24,32)	4	1	20
STS-B	1e-3	100	(32,24)	(32,24)	16	1	20
MNLI	1e-3	100	(24,32)	(32,24)	4	2	20
QNLI	1e-3	100	(24,32)	(32,24)	4	1	20
QQP	1e-3	100	(32,24)	(24,32)	4	1	20

Table 5.12: The hyperparameters of KronA^B experiments are reported in this table. The table is re-used with permission from [Edalati et al., 2022a]. In the original table, the name of STS-B is written SSTS-B, which is corrected here.

KronA$^{\text{B}}_{\text{res}}$ hyperparameters							
Task	learning rate	batch size	$\mathbf{A}_K$	$\mathbf{B}_K$	s	module bias	epoch
CoLA	1e-3	100	(32,24)	(24,32)	16	2	20
RTE	5e-3	100	(32,24)	(24,32)	16	2	20
MRPC	5e-3	100	(32,24)	(24,32)	16	2	20
SST-2	1e-3	100	(32,24)	(24,32)	16	1	20
STS-B	9e-4	100	(32,24)	(32,24)	16	1	20
MNLI	1e-3	100	(32,24)	(24,32)	16	1	20
QNLI	1e-3	100	(32,24)	(24,32)	4	2	20
QQP	1e-3	100	(32,24)	(24,32)	16	1	20

Table 5.13: The hyperparameters of KronA$^{\text{B}}_{\text{res}}$ experiments are reported in this table. The table is re-used with permission from [Edalati et al., 2022a]. In the original table, the name of STS-B is written SSTS-B, which is corrected here.

Chapter 6

Ablation Study

In this chapter, an ablation study related to our experiments for both directions is provided. These experiments and studies helped us to efficiently design our approaches and techniques.

6.1 Experiments Related to KnGPT2

6.1.1 The Importance of Pre-training

We did an experiment to show the importance of pre-training after decomposition. In the first case, the Kronecker decomposed model was fine-tuned and evaluated on MNLI, without pre-training. In the other case, the decomposed model was pre-trained on WikiText-103 before fine-tuning on MNLI. Table 6.1 shows that pre-training the decomposed model, even on a small dataset, significantly improves the performance.

Model	WikiText-103(PPL)	MNLI (f1)
$\text{KnGPT2}_{\text{not pre-trained}}$	28608	69.33
$\text{KnGPT2}_{\text{pre-trained}}$	23.04	**77.97**

Table 6.1: This table shows the effect of pre-training on KnGPT2's performance. The table is drawn based on the results provided in [Edalati et al., 2022b].

6.1.2 Improvement of the ILKD

We utilized a modified version of the ILKD algorithm developed in [Tahaei et al., 2022] to improve the performance of Kronecker Decomposition. In [Tahaei et al., 2022], the attention outputs of all layers of the teacher and student were compared using MSE loss function. However, motivated by [Wang et al., 2020], we only used the output of the last attention block to accelerate the distillation while keeping the performance. In addition, we replaced MSE with K-L divergence as the distance metric [Edalati et al., 2022b, Wang et al., 2020]. Table 6.2 shows the superiority of using K-L divergence over MSE.

Model	CoLA	RTE	MRPC	SST-2	MNLI	QNLI	QQP	Average
KnGPT2 + ILKD$_{MSE}$	41.65	68.95	**88.89**	90.48	80.69	87.66	90.00	78.33
KnGPT2 + ILKD$_{KL}$	**45.36**	**69.67**	87.41	**91.28**	**82.15**	**88.58**	**90.34**	**79.25**

Table 6.2: This table shows the dev score of KnGPT2 that uses K-L divergence or MSE (Equation 3.11) during the fine-tuning. The table is directly re-used with permission from [Edalati et al., 2022b].

Furthermore, in [Tahaei et al., 2022], the CE loss (Equation 3.13) was not used during the pre-training. Therefore, the student was pre-trained only based on the knowledge distilled from the teacher. However, our empirical results (provided in Table 6.3) show that using the CE loss in addition to other losses mentioned in Section 3.5 improves the final performance. Therefore, during both pre-training and fine-tuning, we used Equation 3.13 in our ILKD algorithm.

Model	α_3	CoLA	RTE	MRPC	SST-2	MNLI	QNLI	QQP	Average
KnGPT2 + ILKD	0	40.80	**70.04**	**88.25**	90.71	80.12	87.64	89.64	78.17
KnGPT2 + ILKD	0.1	**45.36**	69.67	87.41	**91.28**	**82.15**	**88.58**	**90.34**	**79.25**

Table 6.3: This table shows the effect of inactivating the CE loss during the pre-training stage on KnGPT2's performance. α_3 is the scale of the CE loss. The table is directly re-used with permission from [Edalati et al., 2022b].

6.2 Experiments Related to the PET Direction

6.2.1 KronA Initialization

We compared two strategies for the initialization of the Kronecker factors in KronA modules. The first strategy used a Normal distribution to initialize $\mathbf{A}_K$ and $\mathbf{B}_K$:

$$\mathbf{A}_K, \mathbf{B}_K \sim \mathcal{N}(0, \frac{1}{d_h})$$

While the second strategy initialized $\mathbf{B}_K$ with a zero vector and $\mathbf{A}_K$ from a Kaming-Uniform (KU) distribution [Hu et al., 2022] where $a = \sqrt{5}$. For the main experiments, we followed the second strategy since as Table 6.4 shows, it achieves better GLUE results.

Init Method	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
$\mathbf{A}_K, \mathbf{B}_K \sim$ Normal	63.36	66.91	91.69	91.97	90.46	86.03	92.33	90.19	84.12
$\mathbf{A}_K \sim$ KU, $\mathbf{B}_K$=0	63.27	77.70	92.52	94.04	91.26	86.03	93.13	90.57	86.06

Table 6.4: The effect of the initialization strategies on the performance of KronA is reported in this table. The table is re-used with permission from [Edalati et al., 2022a]

6.2.2 Step by Step Improvement of KronA^B

Here, the modifications made to a common adapter [Houlsby et al., 2019] to improve it into KronA^B are discussed. Table 6.5 shows the results of applying these modifications. In the first row, Adapter was transformed into the first version of KronA^B by removing the non-linearity and incorporating the Kronecker product instead of the normal matrix multiplication. Note that the first version was applied sequentially on both attention and FFN blocks. In the second row, KronA^B was applied in parallel to the PLM blocks, which improved the performance significantly. In the third row, a scaling factor was added to the module.

Then, we inserted different non-linear activation functions (Mish [Misra, 2020], ReLU [Glorot et al., 2011], GELU [Hendrycks and Gimpel, 2016], and SiLU [Elfwing et al., 2018]) between the Kronecker multiplications (by $\mathbf{A}_K^T$ and $\mathbf{B}_K$ in Equation 3.2) and evaluated it on QNLI to study the effect of non-linearity in our proposed adapter. As Table 6.6 shows, SiLU achieves the best results. However, as the fourth row of Table 6.5 shows, adding SiLU to KronA$^{\text{B}}$ reduces the averaged GLUE score. Also, adding non-linearity increases the latency of KronA$^{\text{B}}$. Consequently, we decided to remove the non-linearity.

Finally, as the last row of Table 6.5 shows, KronA$^{\text{B}}$ achieved the best results when it was removed from attention blocks and applied only to FFNs. Therefore, the final version of KronA$^{\text{B}}$ is applied only in parallel to FFNs while it has a scaling factor instead of a non-linear activation function.

Modification	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
Sequential	15.26	53.28	86.39	87.38	83.78	74.70	84.29	86.63	71.46
Parallel	58.17	69.78	91.58	93.81	90.86	85.68	93.35	90.14	84.17
Parallel+Scale (PS)	62.27	70.50	91.58	94.04	91.01	86.16	93.39	90.61	84.94
PS+SiLU	62.74	69.78	91.89	94.15	90.97	85.98	93.30	90.15	84.87
PS only on FFN	63.74	72.66	92.20	94.72	90.98	85.98	93.12	90.68	85.51

Table 6.5: The effect of each modification applied to KronA$^{\text{B}}$ on its final performance is reported in this table. The table is re-used directly with permission from [Edalati et al., 2022a].

nonlinear function	Mish	ReLU	GELU	GELU$_{\text{new}}$	SiLU
QNLI Performance	93.21	93.28	93.13	93.26	93.30

Table 6.6: This table shows the effect of incorporating different non-linear activation functions on the QNLI score of KronA$^{\text{B}}$. The table is adapted with permission from [Edalati et al., 2022a]. In the original table, the name of activation functions is not capitalized, which is done in this table.

6.2.3 Bounded Scaling of the Residual Connection

We performed an ablation study to choose an optimal method for incorporating the residual connection in KronA$^{\text{B}}_{\text{res}}$. We investigated two strategies based on the possible values for the scale of the residual connection, s_{res}. In one scenario, s_{res} was bounded between zero and one by applying a sigmoid function. This module is called $\text{KronA}^{\text{M}}_{\text{sig-res}}$. In the other scenario, the scale was not bounded, as discussed in Section 4.3. Table 6.7 shows the equations that these two scenarios used to calculate the output in addition to their corresponding GLUE score and latency.

Method	Output Equation	GLUE Score	Latency
$\text{KronA}^{\text{B}}_{\text{res}}$	$\text{KronA}^{\text{B}}_{\text{res}}(\mathbf{X}) = s\mathbf{X}[\mathbf{A}_K \otimes \mathbf{B}_K] + \mathbf{b} + s_{res}\mathbf{X}$	**86.57**	1
$\text{KronA}^{\text{M}}_{\text{sig-res}}$	$\text{KronA}^{\text{M}}_{\text{sig-res}}(\mathbf{X}) = s\mathbf{X}[\mathbf{A}_K \otimes \mathbf{B}_K] + \mathbf{b} + \text{sigmoid}(s_{res})\mathbf{X}$	86.42	1.18

Table 6.7: This table compares the performance of effect of $\text{KronA}^{\text{B}}_{\text{res}}$ and $\text{KronA}^{\text{M}}_{\text{sig-res}}$ to show the effect of bounding S_{res} using a sigmoid function. The table is adapted with permission from [Edalati et al., 2022a]. Compared to the original table, the column "Output Equation" is added to this table. Also, "Avg Score" is replaced with "GLUE Score".

Based on our empirical results, adding the sigmoid function to bound s_{res} increases the latency and reduces the GLUE score. Therefore, we only recommend using $\text{KronA}^{\text{B}}_{\text{res}}$ and discarding $\text{KronA}^{\text{M}}_{\text{sig-res}}$.

Chapter 7

Results and Discussion

In this chapter, Section 7.1 provides the results of our experiments related to using Kronecker Decomposition for Model Compression. Section 7.2 shows how our Kronecker adapters perform compared to the other PET baselines. Finally, a general discussion of our results, limitations, and future works are mentioned in Section 7.3.

7.1 Compression with Kronecker Decomposition

7.1.1 Pre-training Acceleration

	GPT-2$_{\text{base}}$	DistilGPT2	KnGPT2
Parameters[1]	124	82	83
Training time (hrs)	-	> 90[2]	6.5
Dataset size (GB)	40	38	3.2

Table 7.1: This table shows the required time and size of the dataset to pre-train the studied models. The table is re-used with permission from [Edalati et al., 2022b].

[1]The classification head is excluded from the reported number of parameters

[2]Since we did not pre-train DistilGPT2, we did not know the exact pre-training time of this model. DistilBERT has a relatively similar architecture to DistilGPT2 and the size of the pre-training dataset for both models is similar. Therefore, we reported the pre-training time of DistilBERT mentioned in [Sanh et al., 2019]. This number is a reasonable estimation for the pre-training time of DistilGPT2.

As mentioned previously, a brief pre-training is required to calibrate the compressed model weights. Since we used only 10% of OWT, the pre-training of KnGPT2 was significantly faster than the baseline, which is pre-trained on the entire OWT. This is considered an important benefit of KnGPT2 over DistilGPT2. As Table 7.1 demonstrates, using our method reduces the required time and data samples for pre-training, remarkably.

7.1.2 Downstream Results

After the compressed model was calibrated with a brief pre-training, it was fine-tuned on downstream tasks using our proposed ILKD technique.

CLM

WikiText-103 is considered a well-known downstream dataset to evaluate LMs on CLM. We evaluated KnGPT2 on this dataset. As Table 7.2 demonstrates, the Perplexity of KnGPT2 is roughly 3 points lower than DistilGPT2, which means that our model outperforms the baseline with a large margin.

	GPT-2$_{\text{base}}$	DistilGPT2	KnGPT2
PPL	18.8	23.7	20.5

Table 7.2: This table shows the Perplexity of the studied models on the test set of WikiText-103. The table is re-used with permission from [Edalati et al., 2022b].

GLUE

The models were evaluated on the GLUE tasks as well. KnGPT2 was fine-tuned using the ILKD method in one case. In another case, we did not use the ILKD for fine-tuning our model to highlight its effect.

To have a fairer comparison, We also applied the vanilla KD [Hinton et al., 2015, Jafari et al., 2021] to fine-tune the baseline[3]. Tables 7.3 and 7.4 show the performance of the studied models, where the teacher in those experiments that utilized KD was GPT-2_{base}.

Based on our results, KnGPT2 outperforms the baseline by a significant gap. Although using the vanilla KD slightly improved the performance of DistilGPT2, it still fails to compete with KnGPT2. In addition, utilizing the ILKD to fine-tune KnGPT2 significantly improves the performance of our model and makes it roughly similar to the uncompressed model, GPT-2_{small}.

Model	CoLA	RTE	MRPC	SST-2	MNLI	QNLI	QQP	Average
GPT-2_{base}	44.0	63.2	84.5	92.8	81.75	88.7	88.0	77.56
DistilGPT2	32.4	61.9	84.3	90.8	79.55	85.4	87.3	74.52
DistilGPT2 + KD	33	61.5	84.4	90.7	79.85	85.7	87.6	74.67
KnGPT2	36.7	**64.4**	84.5	89.0	78.45	85.6	86.5	75.02
KnGPT2 + ILKD	**41.8**	63.7	**86.5**	**91.5**	**81.6**	**88.4**	**88.5**	**77.42**

Table 7.3: This table shows the GLUE score (test) of the models studied in the Model Compression direction. The table is re-used with permission from [Edalati et al., 2022b].

Model	CoLA	RTE	MRPC	SST-2	MNLI	QNLI	QQP	Average
GPT-2_{base}	47.6	69.31	87.47	92.08	83.12	88.87	90.25	79.81
DistilGPT2	38.7	65.0	87.7	91.3	79.9	85.7	89.3	76.8
DistilGPT2 + KD	38.64	64.98	87.31	89.80	80.42	86.36	89.61	76.73
KnGPT2	37.51	**70.4**	**88.55**	88.64	78.93	86.10	88.87	77
KnGPT2 + ILKD	**45.36**	69.67	87.41	**91.28**	**82.15**	**88.58**	**90.34**	**79.25**

Table 7.4: his table shows the GLUE score (dev) of the models studied in the Model Compression direction. The table is re-used with permission from [Edalati et al., 2022b].

[3]On the one hand, the teacher (GPT-2) and student (DistilGPT2) have a different number of layers. On the other hand, several experiments are required to find the optimal method for using ILKD techniques where the teacher and student have mismatched layers. Therefore, we did not apply our proposed ILKD method to fine-tune DistilGPT2.

7.2 PET with the Kronecker Product

Our proposed Kronecker-based adapters were evaluated on the GLUE tasks and their performance, in terms of the GLUE score, training, and inference time was compared to the SOTA techniques. The discussed methods were utilized to fine-tune T5 with tuning roughly 0.07% of the entire parameters.

KronA and KronA^B have a very similar architecture to LoRA and PA, respectively. Therefore, a comparison between the performance of KronA and LoRA or KronA^B and PA would give us an insightful conclusion about the benefits of incorporating the Kronecker product in adapters.

7.2.1 The GLUE Score

Method	Params[4] (%)	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
FT	100	63.37	74.82	92.73	93.58	90.07	86.16	92.77	91.74	85.65
BitFit	0.12	58.19	68.34	92.58	94.61	90.69	85.73	92.91	90.33	84.17
Adapter	0.07	64.66	71.94	91.27	94.84	90.49	85.91	92.97	90.35	85.30
LoRA	0.07	64.76	74.10	92.10	93.92	91.21	86.08	92.97	90.68	85.73
Compacter	0.07	64.42	76.26	91.52	93.92	91.04	86.14	92.93	90.36	85.82
PA	0.06	64.80	74.10	**93.20**	94.04	91.10	86.24	93.12	90.30	85.86
KronA	0.07	63.27	**77.70**	92.52	94.26	91.30	**86.34**	93.15	90.57	86.14
KronA^B	0.07*	65.74	75.54	92.78	**94.72**	**91.41**	86.22	93.19	**90.68**	86.28
KronA$^B_{res}$	0.07*	**66.73**	76.98	93.15	94.38	91.35	86.20	**93.21**	90.57	**86.57**

Table 7.5: In this table, the GLUE score of our Kronecker-based adapters is compared to the other SOTA techniques for PET. The table is re-used with permission from [Edalati et al., 2022a]. In this table, the column "Params" from the original table is modified to "Params (%)" to clarify that the reported numbers are the percentage of the trainable parameters.

Table 7.5 shows the results of our PET experiments. Generally, in most of the GLUE tasks in addition to the averaged score, all of the proposed Kronecker-based adapters

[4]Please note that KronA^B and KronA$^B_{res}$ might have one or two biases depending on the dev results. Therefore, the number of trainable parameters can be slightly different from the reported number.

outperform the baselines. Also, KronA^B achieves a better score than KronA. Furthermore, KronA$^B_{res}$ outperform all of the studied methods, including KronA^B KronA, which demonstrates the effect of the incorporated residual connection.

7.2.2 Inference Time

Our algorithm to measure the inference time demanded after applying each studied PET method contains the below steps:

- **Step 1**: Generating a dummy input. The sequence length of the dummy input was arbitrarily set to ten.
- **Step 2**: Warming up the GPU by processing the dummy input for 150 iterations[5].
- **Step 3**: Calculating the average time that the model required to process the dummy input for 200 times.
- **Step 4**: Repeating the above steps for three times and reporting the averaged inference time.

The normalized inference latencies are shown in Table 7.6. As expected, BitFit, LoRA, and KronA do not increase the inference latency. Therefore, KronA could be an appropriate candidate for latency-sensitive applications. However, PA is slightly faster than KronA^B. In addition, KronA$^B_{res}$ is around 10% slower than KronA^B due to the extra residual connection.

Method	FT	LoRA	KronA	BitFit	Adapter	PA	Compacter	KronA^B	KronA$^B_{res}$
Inference Latency(%)	100	100	100	100	146	113	181	127	136

Table 7.6: The inference time demanded by each method is normalized and shown in this table. The table is adapted with permission from [Edalati et al., 2022a]. The third row of the original table is removed in this table.

[5]To measure the accurate inference time that a model requires to process an input sample on a GPU, the GPU must be warmed up first to exclude the required time for activation and initialization of the GPU.

7.2.3 Training Time

Please note those hyperparameters that affect the training time (e.g, batch size, epochs, and gradient accumulation steps) were fixed in our experiments for all of the PET methods. Therefore, the comparison between the training speed up of the methods was fair. Finally, the training time of each method on the GLUE tasks is normalized and shown in table 7.7.

By comparing the training time of KronA and LoRA or KronA^B and PA, we can conclude that using the Kronecker product marginally increases the training time. Similar to the inference time, adding the residual connection increases the training time of KronA$^B_{res}$. Generally, our Kronecker-based adapters are marginally slower than the baselines. However, the gap is not significant and using our proposed methods notably reduces the training time, compared to FT.

Method	CoLA	RTE	MRPC	SST-2	STS-B	MNLI	QNLI	QQP	Avg
FT	1	1	1	1	1	1	1	1	1
BitFit	0.58	0.64	0.65	0.66	0.66	0.65	0.64	0.62	0.64
Adapter	0.82	0.71	0.72	0.78	0.76	0.72	0.69	0.69	0.73
LoRA	0.79	0.69	0.7	0.76	0.72	0.7	0.68	0.68	0.72
KronA	0.8	0.72	0.75	0.81	0.77	0.74	0.73	0.73	0.75
Compacter	0.88	0.74	0.78	0.86	0.81	0.75	0.74	0.76	0.79
PA	0.7	0.78	0.81	0.73	0.7	0.67	0.66	0.65	0.71
KronA^B	0.84	0.7	0.72	0.79	0.75	0.72	0.7	0.71	0.74
KronA$^B_{res}$	0.91	0.85	0.81	0.91	0.76	0.78	0.73	0.74	0.81

Table 7.7: The training time demanded by each method is normalized and shown in this table. The table is re-used with permission from [Edalati et al., 2022a].

7.3 General Discussion

7.3.1 Key Findings

First, we show that an auto-regressive LM like GPT-2 can be compressed by decomposing its weight matrices into smaller matrices using Kronecker Decomposition. Also, a relatively brief pre-training is sufficient to retrieve the lost information during Kronecker Decomposition. In addition, the proposed ILKD from the base model to the compressed model plays a crucial role in performance improvement.

Second, our results in Table 7.5 illustrate that using the Kronecker product instead of the low-rank projections, improves the performance of the PET approaches. Based on Tables 7.7 and 7.6, although using the Kronecker product slightly increases the training time, it still reduces the training time compared to FT and some of the baselines such as Compacter, significantly. The mentioned point is valid during the inference phase except for KronA since it does not increase the inference latency.

7.3.2 Interpretation of the Findings

Our findings for the compression direction mean that an efficient and powerful compression technique in terms of the pre-training time and performance is developed by combining Kronecker Decomposition and the ILKD. This compression technique can effectively compress LMs while maintaining their performance. Since this technique decomposes linear weight matrices, it could be theoretically utilized to compress every neural network composed of linear layers.

Also, our PET results mean that it is possible to efficiently fine-tune a PLM on a downstream task by tuning only the Kronecker-based adapters inserted in the specific blocks of the model. Although the parameters of these adapters are negligible compared to the huge size of a PLM, the Kronecker-based adapters perform better than the traditional low-rank adapters on the downstream tasks. Also, at the expense of more latency, we can improve the performance of the Kronecker-based adapters by adding a learnable residual connection where applicable. The Kronecker-based adapters can meet our performance expectations, especially when a slight increase in training or inference time is tolerable.

7.3.3 Limitations

The performance of the Kronecker-based layers, in terms of accuracy and latency, is sensitive to the shape of the Kronecker factors. Therefore, it is needed to find the optimal shapes. In our analysis, we only tuned the Kronecker factor shapes of the KronA modules while the shape of KronA^{B} and $\text{KronA}^{\text{B}}_{\text{res}}$ was fixed across different tasks. Therefore, our results can be sub-optimal. Finding the optimum shape per task and investigating the computation overhead through hyperparameter search is an area for future work.

The proposed compression technique is not compared with other techniques like Pruning or Quantization since those were not applied to our target model, GPT-2. Therefore, our conclusion for the superiority of Kronecker Decomposition needs further experiments to be proven. One might ask about the reason for choosing GPT-2. The answer is that we wanted to apply our method to a model that is not already compressed efficiently.

In addition, due to time and resource limitations, we only applied Kronecker Decomposition to the smallest version of GPT-2. However, more experiments on larger models and compression ratios are needed for a more solid conclusion.

Theoretically, Kronecker Decomposition could be applied to every neural network with linear layers. However, more experiments are needed to confirm it practically.

Application of the proposed Kronecker adapters is limited to the acceleration of the fine-tuning stage. They are not recommended to be utilized for the pre-training stage since their limited representation power should not be sufficient to capture the general domain pre-training information.

Further experiments are needed to measure the occupied dynamic memory of the Kronecker adapters compared to the other baselines.

Due to resource limitations, we did not use Equation 3.2 to efficiently calculate the Kronecker layers' output. Therefore, the compressed model might not gain any inference speed-up or FLOPS reduction.

7.3.4 Future Works

To generate a fixed-size matrix, **W**, by the Kronecker product of two smaller matrices, usually, there are several options for the shape of the Kronecker factors. In our work, we noticed that the shape of the Kronecker factors can affect the final results significantly. To the best of our knowledge, the effect of the Kronecker factors shape on the performance of the Kronecker product is not studied yet. One potential next step for our work could be a study on the effect of the Kronecker factors shape on the performance to develop an algorithm for finding the optimal shapes. This study can be applied to improve the performance of our proposed methods as well as other works that exploit the Kronecker product.

Based on [Hameed et al., 2022], summation can improve the performance of Kronecker Decomposition for compression of the convolutional layers. Also, the summation is used in Compacter [Karimi Mahabadi et al., 2021] for PET. Summation means adding the Kronecker product of several pairs of the Kronecker factors,

$$\sum_{n \in [1:N]} \mathbf{A}_{K_n} \otimes \mathbf{B}_{K_n}$$

instead of using one pair of the Kronecker factors, $\mathbf{A}_K \otimes \mathbf{B}_K$. Adding the summation might improve the performance of our proposed methods. However, we did not study the summation effect due to time limitations; thus, it can be a promising future direction.

Our results show that Kronecker Decomposition can successfully compress GPT-2. Also, it can easily be applied to other Transformer-based models. Nevertheless, the ability of Kronecker Decomposition to achieve good results for the compression of larger models, which require larger compression rates, is an open question.

While most of the mentioned baselines and our approaches for PET achieve comparable results to the fine-tuning on in-domain data, the robustness of models trained by these techniques on out-of-domain data and adversarial attacks is not studied yet. On the one hand, using the PET methods might increase the robustness of models since they train a model by using fewer parameters compared to FT. Training by fewer parameters for a downstream task that often uses a relatively small dataset, may prevent over-fitting. Consequently, the performance on a new domain data is improved. On the other hand, some other works [Brown et al., 2020, Du et al., 2023] show that by decreasing the number of trainable parameters, the ability of PLMs to generalize on a new domain data decreases. Therefore, this is an interesting line of research that we could not follow due to time limitations.

Some recent works have used a unified framework to combine the existing PET baselines to achieve SOTA results [He et al., 2022a, Mao et al., 2022, Zhang et al., 2022]. Inspired by the mentioned papers, combining the proposed methods in our work for PET might significantly improve the performance.

Finally, the mentioned approaches for PET in our work can be applied to other fields of Machine Learning, where Transformer-based models are used such as Computer Vision [He et al., 2022b], Speech Recognition, and Time-Series Prediction.

Chapter 8

Conclusion

In this work, improving the efficiency of NLP models by using the Kronecker product is studied. First, some of the challenges regarding the ever-growing size of NLP models, which are time and resource consuming fine-tuning and difficulty of deployment on edge devices, are mentioned. Then, two research directions for the efficient NLP are explained. Also, the methodology of existing baselines and SOTA methods in addition to their strengths and weaknesses, are discussed.

The first studied direction is Model Compression, which enables reducing the size of PLMs while keeping their performance. We have developed a compression technique, called Kronecker Decomposition, by combining Matrix Decomposition and KD as two existing approaches for Model Compression. Our technique reduces the size of the weight matrices in linear layers of a model. In this direction, we have focused on the compression of GPT-2. Nevertheless, our proposed approach for GPT compression can easily be applied to every model that is composed of linear layers.

Therefore, we compress GPT-2 into KnGPT2 by Kronecker Decomposition. Then, KnGPT2 is pre-trained on a relatively small dataset for one epoch to retrieve the lost knowledge during the decomposition process. Pre-training of KnGPT2 is significantly faster than its rival, DistilGPT2. However, KnGPT2 outperforms DistilGPT2 when both are fine-tuned on the downstream datasets such as WikiText-103 and GLUE. We also have introduced an ILKD method, which is used for both pre-training and fine-tuning of KnGPT2 to improve its performance.

Second, we have focused on the PET methods, which aim to make the fine-tuning process more efficient in terms of the required time or resources while keeping the performance. In this direction, we have proposed to replace the common matrix multiplications of the down and up projection weight matrices in adapters with the Kronecker product. Therefore, by removing the low-rank projections, the rank deficiency problem of the mentioned methods is solved, which improves the downstream results.

We have developed KronA and KronA^{B} as the updated versions of LoRA and PA, respectively. We also have suggested adding a residual connection scaled by a learnable factor to KronA^{B} to further improve the fine-tuning results at the expense of less speed-up during both the training and inference phases.

Our proposed PET methods outperform the other SOTA baselines on the GLUE benchmark. Also, the speed-up of our proposed methods is comparable to the other baselines. Consequently, our methods significantly reduce the required training time compared to FT.

Bibliography

E. Agirre, L. M'arquez, and R. Wicentowski, editors. *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*. Association for Computational Linguistics, Prague, Czech Republic, June 2007.

L. R. Bahl, F. Jelinek, and R. L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2):179–190, 1983. doi: 10.1109/TPAMI.1983.4767370.

H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King. BinaryBERT: Pushing the limit of BERT quantization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4334–4348, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.334.

R. Bar Haim, I. Dagan, B. Dolan, L. Ferro, D. Giampiccolo, B. Magnini, and I. Szpektor. The second PASCAL recognising textual entailment challenge, 2006.

E. Ben Zaken, Y. Goldberg, and S. Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

L. Bentivogli, B. Magnini, I. Dagan, H. T. Dang, and D. Giampiccolo. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of the Second Text Analysis Conference, TAC 2009, Gaithersburg, Maryland, USA, November 16-17, 2009*. NIST, 2009.

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. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

L. Chen, Y. Chen, J. Xi, and X. Le. Knowledge from the original network: restore a better pruned network with knowledge distillation. *Complex & Intelligent Systems*, 8:709–718, 01 2021. doi: 10.1007/s40747-020-00248-y.

D. Chicco, M. J. Warrens, and G. Jurman. The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment. *IEEE Access*, 9:78368–78381, 2021. doi: 10.1109/ACCESS.2021.3084050.

P. Clarkson and T. Robinson. Towards improved language model evaluation measures. In *Proc. 6th European Conference on Speech Communication and Technology (Eurospeech 1999)*, pages 1927–1930, 1999. doi: 10.21437/Eurospeech.1999-423.

I. Dagan, O. Glickman, and B. Magnini. The pascal recognising textual entailment challenge. In J. Quiñonero-Candela, I. Dagan, B. Magnini, and F. d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-33428-6.

D. Demszky, K. Guu, and P. Liang. Transforming question answering datasets into natural language inference datasets. *arXiv preprint arXiv:1809.02922*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.

Y. Dodge. *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008. ISBN 978-0-387-32833-1. doi: 10.1007/978-0-387-32833-1_379.

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

M. Du, S. S. Mukherjee, Y. Cheng, M. Shokouhi, X. Hu, and A. H. Awadallah. What do compressed large language models forget? robustness challenges in model compression. In *17th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2023)*, April 2023.

A. Edalati, M. Tahaei, I. Kobyzev, V. P. Nia, J. J. Clark, and M. Rezagholizadeh. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*, 2022a.

A. Edalati, M. Tahaei, A. Rashid, V. Nia, J. Clark, and M. Rezagholizadeh. Kronecker decomposition for GPT compression. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 219–226, Dublin, Ireland, May 2022b. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.24.

S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks : the official journal of the International Neural Network Society*, 107:3–11, 2018.

D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 201–208, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

D. Giampiccolo, B. Magnini, I. Dagan, and B. Dolan. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics, 2007.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

A. Gokaslan and V. Cohen. Openwebtext corpus, 2019.

Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

M. Gordon, K. Duh, and N. Andrews. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 143–155, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.repl4nlp-1.18.

J. Gou, B. Yu, S. J. Maybank, and D. Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

E. Grave, A. Joulin, and N. Usunier. Improving neural language models with a continuous cache. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

M. G. A. Hameed, M. S. Tahaei, A. Mosleh, and V. Partovi Nia. Convolutional neural network compression through generalized kronecker product decomposition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(1):771–779, Jun. 2022. doi: 10.1609/aaai.v36i1.19958.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig. Towards a unified view of parameter-efficient transfer learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022a.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.123.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90.

X. He, C. Li, P. Zhang, J. Yang, and X. E. Wang. Parameter-efficient fine-tuning for vision transformers. *arXiv preprint arXiv:2203.16329*, 2022b.

Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1398–1406. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.155.

Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 2234–2240. AAAI Press, 2018. ISBN 9780999241127.

H. V. Henderson, F. Pukelsheim, and S. R. Searle. On the history of the kronecker product. *Linear & Multilinear Algebra*, 14:113–120, 1983.

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

D. Hendrycks, K. Lee, and M. Mazeika. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*, pages 2712–2721. PMLR, 2019.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

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 K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

Z. Huang, Y. Zou, V. Bhagavatula, and D. Huang. Comprehensive attention self-distillation for weakly-supervised object detection. In *Proceedings of the 34th Interna-*

tional Conference on Neural Information Processing Systems, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

A. Jafari, M. Rezagholizadeh, P. Sharma, and A. Ghodsi. Annealing knowledge distillation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2493–2504, Online, Apr. 2021. Association for Computational Linguistics.

X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.372.

K. Jing and J. Xu. A survey on neural network language models. *arXiv preprint arXiv:1906.03591*, 2019.

M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

R. Karimi Mahabadi, J. Henderson, and S. Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoust. Speech Signal Process.*, 35:400–401, 1987.

J. Kim, M. Hyun, I. Chung, and N. Kwak. Feature fusion for online mutual knowledge distillation. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4619–4625, 2021. doi: 10.1109/ICPR48806.2021.9412615.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703.

S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

T. Li, Y. E. Mesbahi, I. Kobyzev, A. Rashid, A. Mahmud, N. Anchuri, H. Hajimolahoseini, Y. Liu, and M. Rezagholizadeh. A short study on compressing decoder-based language models. *arXiv preprint arXiv:2110.08460*, 2021.

X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353.

T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.07.045.

V. Lioutas, A. Rashid, K. Kumar, M. A. Haidar, and M. Rezagholizadeh. Improving word embedding factorization for compression using distilled nonlinear neural decomposition. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2774–2784, 2020.

H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965. Curran Associates, Inc., 2022.

P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), jan 2023. ISSN 0360-0300. doi: 10.1145/3560815.

Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020. doi: 10.1162/tacl_a_00343.

Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao. Post-training quantization for vision transformer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 28092–28103. Curran Associates, Inc., 2021.

Y. Mao, L. Mathias, R. Hou, A. Almahairi, H. Ma, J. Han, S. Yih, and M. Khabsa. UniPELT: A unified framework for parameter-efficient language model tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6253–6264, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.433.

B. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975. ISSN 0005-2795. doi: https://doi.org/10.1016/0005-2795(75)90109-9.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010*, volume 2, pages 1045–1048, 01 2010.

T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531, 2011a. doi: 10.1109/ICASSP.2011.5947611.

T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. H. Cernocky. Rnnlm - recurrent neural network language modeling toolkit. In *IEEE Automatic Speech Recognition and Understanding Workshop*, December 2011b.

D. Misra. Mish: A self regularized non-monotonic activation function. In *British Machine Vision Conference*, 2020.

T. R. Niesler and P. C. Woodland. A variable-length category-based n-gram language model. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings (ICASSP)*, volume 1, pages 164–167. IEEE, 1996.

N. Passalis and A. Tefas. Learning deep representations with probabilistic knowledge transfer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 268–284, 2018.

P. Passban, Y. Wu, M. Rezagholizadeh, and Q. Liu. ALP-KD: attention-based layer projection for knowledge distillation. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelli-*

gence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 13657–13665. AAAI Press, 2021.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

G. Prato, E. Charlaix, and M. Rezagholizadeh. Fully quantized transformer for machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1–14, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.1.

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

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

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

A. Rashid, V. Lioutas, and M. Rezagholizadeh. MATE-KD: Masked adversarial TExt, a companion to knowledge distillation. In *Proceedings of the 59th Annual Meeting of the*

Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 1062–1071, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.86.

J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988. doi: 10.1080/00031305.1988.10475524.

R. Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, 2000. doi: 10.1109/5.880083.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

S. Shen, D. Zhen, J. Ye, L. Ma, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821, 04 2020. doi: 10.1609/aaai.v34i05.6409.

J. R. Shoenfield. Markov a. a.. theory of algorithms. english translation by schorr-kon jacques j., and program for scientific translations staff. published for the national science foundation, washington, d.c., and the department of commerce by the israel program for scientific translations, jerusalem 1961, 444 pp. *Journal of Symbolic Logic*, 27(2): 244–244, 1962. doi: 10.2307/2964158.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. ISSN 00029556.

Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.195.

M. Sundermeyer, R. Schlüter, and H. Ney. LSTM neural networks for language modeling. In *Proc. Interspeech 2012*, pages 194–197, 2012. doi: 10.21437/Interspeech.2012-65.

M. Sundermeyer, H. Ney, and R. Schlüter. From feedforward to recurrent lstm neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):517–529, 2015. doi: 10.1109/TASLP.2015.2400218.

M. Tahaei, E. Charlaix, V. Nia, A. Ghodsi, and M. Rezagholizadeh. KroneckerBERT: Significant compression of pre-trained language models through kronecker decomposition and knowledge distillation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2116–2127, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.154.

U. Thakker, J. Beu, D. Gope, C. Zhou, I. Fedorov, G. Dasika, and M. Mattina. Compressing rnns for iot devices by 15-38x using kronecker products. *arXiv preprint arXiv:1906.02876*, 2019.

U. Thakker, P. Whatamough, M. Mattina, and J. Beu. Compressing language models using doped kronecker products. *arXiv preprint arXiv:2001.08896*, 2020.

C. F. Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

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

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5776–5788. Curran Associates, Inc., 2020.

A. Warstadt, A. Singh, and S. R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019. doi: 10.1162/tacl_a_00290.

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

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. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. 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. doi: 10.18653/v1/2020.emnlp-demos.6.

J.-N. Wu. Compression of fully-connected layer in neural network by kronecker product. In *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, pages 173–179, 2016. doi: 10.1109/ICACI.2016.7449822.

S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

K. Xu, L. Rui, Y. Li, and L. Gu. Feature normalized knowledge distillation for image classification. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV*, page 664–680, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58594-5. doi: 10.1007/978-3-030-58595-2_40.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. 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.

J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7130–7138, 2017. doi: 10.1109/CVPR.2017.754.

X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76, 2017. doi: 10.1109/CVPR.2017.15.

O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39, 2019. doi: 10.1109/EMC2-NIPS53020.2019.00016.

L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3712–3721, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society. doi: 10.1109/ICCV.2019.00381.

T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.

W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.37.

Z. Zhang, W. Guo, X. Meng, Y. Wang, Y. Wang, X. Jiang, Q. Liu, and Z. Yang. Hyperpelt: Unified parameter-efficient language model tuning for both language and vision-and-language tasks. *arXiv preprint arXiv:2203.03878*, 2022.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.