Kronecker Decomposition for GPT Compression

Ali Edalati[2], **Marzieh Tahaei**[1], **Ahmad Rashid**[1],
Vahid Partovi Nia[1], **James J. Clark**[2], **Mehdi Rezagholizadeh**[1]
[1] Huawei Noah Ark Lab
[2] McGill University
ali.edalati@mail.macgill.ca

Abstract

GPT is an auto-regressive Transformer-based pre-trained language model which has attracted a lot of attention in the natural language processing (NLP) domain due to its state-of-the-art performance in several downstream tasks. The success of GPT is mostly attributed to its pre-training on huge amount of data and its large number of parameters (from $\sim$100M to billions of parameters). Despite the superior performance of GPT (especially in few-shot or zero-shot setup), this overparameterized nature of GPT can be very prohibitive for deploying this model on devices with limited computational power or memory. This problem can be mitigated using model compression techniques; however, compressing GPT models has not been investigated much in the literature. In this work, we use Kronecker decomposition to compress the linear mappings of the GPT-2 model. Our Kronecker GPT-2 model (KnGPT2) is initialized based on the Kronecker decomposed version of the GPT-2 model and then is undergone a very light pre-training on only a small portion of the training data with intermediate layer knowledge distillation (ILKD). Finally, our KnGPT2 is fine-tuned on down-stream tasks using ILKD as well. We evaluate our model on both language modeling and General Language Understanding Evaluation benchmark tasks and show that with more efficient pre-training and similar number of parameters, our KnGPT2 outperforms the existing DistilGPT2 model significantly.

1 Introduction

Recently, development and deployment of pre-trained language models (PLMs) has improved the performance of NLP models significantly [2, 18, 29, 21, 18]. PLMs are mostly Transformer-based models, which are pre-trained on enormous unlabeled data. Although Transformer-based PLMs are powerful in performance, their huge size is a barrier for efficient training or inference of these models on lower capacity devices with 3 memory, computation and energy constraints. Therefore, there has been a growing volume of literature focused on developing frameworks for compressing these large PLMs. 3

Like other deep learning models, the main directions of model compression for PLMs are using following methods in isolation or combination: low-bit quantization [5, 17], pruning [6], knowledge distillation (KD) [8] and matrix decomposition ([30, 12]).

PLMs can be divided into encoder-based and auto-regressive models such as the BERT [2, 14] and GPT [1] family respectively. Although the size of BERT family models is usually smaller than the GPT family, compressing the BERT family has been investigated much more in the literature (e.g. DistilBERT [20], TinyBERT [10], MobileBERT [22], ALP-KD [16], MATE-KD [19], Annealing-KD [9] and BERTQuant [31]). On the other hand, to the best of our knowledge, the GPT family has

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

barely a handful of compressed models, among them the DistilGPT2[1] model is very prominent. The DistilGPT2 model is heavily pre-trained for 3 epochs on the large OpenWebText dataset[2]. Moreover, it is evident in the literature that the GPT model cannot compete with BERT on natural language understanding (NLU) tasks [13]. Therefore, developing an efficient compressed GPT model with comparable NLU performance is still an open problem.

In this paper, we use Kronecker decomposition, which has been recently used for BERT compression [23], for compression of the GPT-2 model (we refer to our model as KnGPT2 in this paper). We use Kronecker decomposition to represent the weight matrices of linear layers in GPT-2 by smaller matrices which can reduce the size and computation overhead. We use Kronecker decomposition to compress the embedding and Transformer layers of GPT-2. For Transformer layers, the linear layers of multi-head attention (MHA) and the feed-forward network (FFN) blocks of Transformer layers are decomposed into Kronecker layers.

Kronecker decomposition leads to reduction in expressiveness of the model. We use a very light pre-training with intermediate layer knowledge distillation (ILKD) to address this issue, which improves the performance of the compressed model significantly. It is worth mentioning that for our pre-training, we use $1/10^{\text{th}}$ of the DistilGPT2's pre-training data (i.e. OpenWebText) only for 1 epoch (instead of 3 epochs in DistilGPT2). Furthermore, in this paper, our framework is applied to GPT-2 but it can be easily exploited to compress other models as well. To summarize contributions of this paper, we mention the following points:

- To the best of our knowledge, we are the first work which uses Kronecker decomposition for compression of the GPT model.
- Our KnGPT2 model improves training efficiency and performance of the DistilGPT2 model significantly.
- We evaluate the performance of our KnGPT2 on both language modeling and the GLUE benchmark tasks.

2 Related Works

In this section, first, we review some of previous works that have deployed Kronecker decomposition for compression of deep learning models. Then, some works related to GPT compression are covered. [32] is the first work that used summation of multiple Kronecker products to compress the weight matrices in fully-connected networks and small convolutional neural networks. [24] proposed a hybrid method which separates the weight matrices into an upper and a lower part, upper part remains untouched but the lower part decomposes to Kronecker products. They used this approach for small language models to be utilized on internet of things (IoT) applications. Recently, [25] extended the mentioned hybrid method to non-IoT applications by adding a sparse matrix to the Kronecker products. [23] has deployed a similar approach to ours to compress BERT which achieved promising results but to the best of our known, this work is the first attempt for GPT compression using Kronecker decomposition.

DistilGPT2 [3] is one of the most successful and well-known compressed versions of GPT-2 which is considered as a baseline in this paper. DistilGPT2 has 82M parameters compared to 124M parameters for GPT-2_{Small} and is trained using KD on OpenWebTextCorpus which is a reproduction of OpenAI's WebText dataset.

3 Methodology

In this section, we first provide some background on Kronecker product and its mathematical properties. We then explain how Kronecker factorization can be used for the compression of linear layers and subsequently for the GPT model.

[1] https://transformer.huggingface.co/model/distil-gpt2

[2] https://huggingface.co/datasets/openwebtext

[3] For further details, see https://huggingface.co/distilgpt2

3.1 Kronecker Product

The Kronecker product is a matrix operation (denoted by $\otimes$) which takes two matrices as input and generates a block matrix as output. Assume that $\mathbf{A}$ is a matrix $\in \mathbb{R}^{m_1 \times n_1}$ and $\mathbf{B}$ is a matrix $\in \mathbb{R}^{m_2 \times n_2}$, $\mathbf{A} \otimes \mathbf{B}$ is equal to a block matrix $\in \mathbb{R}^{m \times n}$, where $m = m_1 m_2$, $n = n_1 n_2$ and each block (i, j) is obtained by multiplying element a_{ij} by matrix $\mathbf{B}$.

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

Kronecker product has attractive abstract algebraic properties such as

$$\mathbf{A} \otimes (\mathbf{B} + \mathbf{C}) = \mathbf{A} \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{C}, \quad (\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}, \quad (\mathbf{A} \otimes \mathbf{B})^{\top} = \mathbf{A}^{\top} \otimes \mathbf{B}^{\top},$$

for more details see [7]. The interesting properties of the Kronecker product makes it an attractive tool for decomposition of large matrices. The Kronecker product is also a flexible method to simplify the notation of large block matrices, both in linear mixed effect models and multilevel models [4]. It is also a well-known technique to represent large repetitive structured graphs using the Kronecker product [11]. One of the most important characteristics of a matrix is its determinant and it is well-known that for two square matrices $\mathbf{A}$ and $\mathbf{B}$ of size n, and m, $|\mathbf{A} \otimes \mathbf{B}| = |\mathbf{A}|^n |\mathbf{B}|^m$. This property explains the superiority of Kronecker compared to the other decomposition methods for large matrices. By choosing the right n and m, a large matrix $\mathbf{W} = \mathbf{A} \otimes \mathbf{B}$ can be decomposed to much smaller matrices such that the above determinant equation holds.

3.2 GPT-2 Compression using Kronecker Factorization

We can represent a weight matrix, $\mathbf{W} \in \mathbb{R}^{m \times n}$, by two smaller matrices, $\mathbf{A} \in \mathbb{R}^{m_1 \times n_1}$ and $\mathbf{B} \in \mathbb{R}^{m_2 \times n_2}$ such that $\mathbf{W} = \mathbf{A} \otimes \mathbf{B}$ and $m = m_1 m_2$, $n = n_1 n_2$. This leads to reduction in the number of parameters from mn for the original matrix to $m_1 n_1 + m_2 n_2$ for the Kronecker factorized version. For example, assume that size of $\mathbf{W}$ is 1024×1024, we can represent it by two matrices of sizes 512×512 and 2×2 for which the compression factor will be roughly equal to 4.

In the following we explain how this work uses Kronecker factorization for compression of the GPT-2 model. In large language models, embedding layer usually takes a large portion of the memory. Let $\mathbf{W}^E \in \mathbb{R}^{v \times d}$ be the lookup table for the input embedding where v is the vocabulary size and d is the embedding dimension. To compress the embedding layer using Kronecker decomposition we use the same method as in [23]. We define $\mathbf{A}^E \in \mathbb{R}^{v \times d/f}$ and $\mathbf{B}^E \in \mathbb{R}^{1 \times f}$, where f is a factor of d. There are two reasons for this decision: first, similar to $\mathbf{W}^E$, in the $\mathbf{A}^E$ matrix every row will indicate embedding of a single word. Second, the embedding of each word, E_i, can be obtained by $\mathbf{A}_i^E \otimes \mathbf{B}$ therefore the computation complexity of this operation is $\mathcal{O}(d)$ which is very efficient.

The transformer architecture is composed of N identical layers each having MHA followed by FFN. In the MHA module, there are linear layers which calculate the Query, Key and Value by multiplying the input vector by $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$, respectively. Also, in the FFN module, there are two fully connected layers that can be represented as $\mathbf{W}^{c_{\text{fc}}} and \mathbf{W}^{c_{\text{proj}}}$. In this work, all of the mentioned weight matrices at different heads and layers of the transformer are decomposed into Kronecker factors.

For initialization, similar to [23], the Kronecker factors $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ are estimated from the corresponding weight matrix $\mathbf{W}$ in the original uncompressed pre-trained model using the solution to the nearest Kronecker problem

$$(\hat{\mathbf{A}}, \hat{\mathbf{B}}) = \underset{(\mathbf{A},\mathbf{B})}{\operatorname{argmin}} \|\mathbf{W} - \mathbf{A} \otimes \mathbf{B}\|_2^2 \,.$$

The solution to this optimization can be found by rank-1 singular value decomposition (SVD) approximation of the reshaped $\mathbf{W}$, see [26] for details.

3.3 Knowledge Distillation

In this section, the knowledge distillation method used for training the KnGPT2 model is explained. The same method is used in the pre-training and fine-tuning stages.

Model	Embedding	Q,K,V	FFN*
GPT-2$_{\text{Small}}$	50527×768	768×768	3072×768
DistilGPT2	50527×768	768×768	3072×768
KnGPT2	$A : 50527 \times 384$, $B : 1 \times 2$	$A : 384 \times 768$, B:2×1	$A : 1536 \times 768$, $B : 2 \times 1$

Table 1: This table shows configuration of the models. Note that FFN block has two projections that shape of one is the transpose of the other one and here, only shape of one of them is mentioned. Also, for KnGPT2, mentioned shapes for transformer layer belong to half of the layers that are decomposed -layers with odd numbers- and shape of the other half are the same with the GPT-2 model.

Phase	Batch size	Learning rate	α_1	α_2	α_3	α_4
Pre-training	1	0.00025	0.5	0.5	0.5	0.1
Fine-tuning	16	2e-5	0.5	0.5	0.5	0.02

Table 2: hyper-parameters that are used for pre-training and fine-tuning.

Let T and S represent the teacher model, GPT-2, and the student model, KnGPT2, respectively. For a batch of data $(\mathbf{x}, \mathbf{y})$, E^S and E^T are outputs of the embedding layers of the student and teacher models respectively. Also, Att_l^S and Att_l^T are the attention distributions obtained by applying softmax on the scaled dot product between query and key. H_l^S and H_l^T are the hidden state outputs of the layer l. Note that by using the Kronecker factorization, like other decomposition methods, the number of layers and dimensions of the output matrices in the student model remain intact so we can directly obtain the difference of output of a specific layer in student an teacher model without the need for projection. For the embedding layer we use the mean squared error (MSE) between the teacher's and student's embeddings:

$$L_{\text{Embedding}}(x) = \text{MSE}\{E^S(x), E^T(x)\} \tag{2}$$

For the MHA modules, similar to [28], we use Kullback–Leibler divergence (KL) between the attention distributions of the student and the teacher.

$$L_{\text{Attention}}(x) = \sum_l \text{KL}\{\text{Att}_l^S(x), \text{Att}_l^T(x)\} \tag{3}$$

For the FFN modules, we simply use the MSE between the output of the second fully connected layer in the student and teacher:

$$L_{\text{Hidden States}}(x) = \sum_l \text{MSE}\{H_l^S(x), H_l^T(x)\} \tag{4}$$

The final loss is calculated a linear combination of the above losses as well as the cross entropy loss.

$$\text{Loss}(x, y) = \sum_{(x,y)} \alpha_1 L_{\text{Embedding}}(x) + \alpha_2 L_{\text{Attention}}(x) + \alpha_3 L_{\text{Hidden States}}(x) + \alpha_4 L_{\text{Cross Entropy}}(x, y) \tag{5}$$

After decomposing the teacher model, GPT-2, into KnGPT2, the performance of the model drops significantly. This drop is mainly because of the approximation of linear weight matrices using the corresponding Kronecker factors. Therefore, pre-training of the compressed model on a small corpus for a few epochs is necessary to retrieve the information which are lost during decomposition. Inspired by [10], we pre-trained the model on a small portion, 10%, of the OpenWebText dataset [3] for one epoch and we used the KD method which is discussed in Section 3.3 to improve the performance of the compressed model.

4 Experiments

We evaluated our proposed algorithm, KnGPT2, on language modeling and text classification. For language modeling we use the Wikitext-103 [15] dataset.For classification we use seven of the classification tasks of the General Language Understanding Evaluation (GLUE) benchmark [27]. These datasets can be broadly divided into 3 families of problems. Single set tasks which include linguistic acceptability (CoLA) and sentiment analysis (SST-2), similarity and paraphrasing tasks (MRPC and QQP), and inference tasks which include Natural Language Inference (MNLI and RTE) and Question Answering (QNLI).

	GPT-2_{Small}	DistilGPT2	KnGPT2
Parameters*	124	82	83
Training time (hrs)	-	>90[4]	6.5
Dataset size (GB)	40	38	3.2

Table 3: Training details for GPT-2 compression. Note that number of parameters of the models are reported excluding the output embedding layer in language modelling which is not compressed, is equal to row Parameters*

	GPT-2_{Small}	DistilGPT2	KnGPT2
Perplexity	18.8	23.7	20.5

Table 4: Test Perplexity on WikiText-103.

4.1 Experimental Setup

The KnGPT2 model is compressed from the GPT-2_{Small} [18] model. GPT-2_{Small} is 124 million parameters. Our baseline is DistilGPT2 which has about 82 million parameters so our KnGPT2 model is compressed to the same size (83 million parameters) for a fair comparison. To achieve this, we compress half the layers of transformer block (odd numbered ones) in addition to the embedding layer by a factor of 2. Table 1 shows the configuration of the models. Table 2 shows hyper-parameters that are used for pre-training and fine-tuning.

4.2 Pre-training

After the base model is compressed using Kronecker decomposition, performance of the compressed model drops significantly since the weight matrices with the Kronecker factors are approximate. Pre-training on a relatively small data set for one epoch helps in retrieving the accuracy. Therefore, KnGPT2 is pre-trained on 10% of OpenWebText which is 10 times less the DistiGPT2 model. As shown on Table 3 the training time for KnGPT2 is much faster as well.

4.3 Results

We measure the performance of our compressed model on two tasks. First we evaluate on language modeling using the Wikitext-103 dataset. The initialized models a are first trained on this dataset and then evaluated on the provided test set. The results are shown on Table 4. Although the DistilGPT2 is pre-trained longer and on a larger dataset the KnGPT2 achieves a lower perplexity.

Next the performance of the models is evaluated on both the development (Table 5) and test (Table 6) sets of seven datasets of the GLUE benchmark. In addition to employing the cross-entropy loss for fitting the labels we also experiment with KD. For DistilGPT, we apply the basic KD algorithm also referred to in the literature as Vanilla KD [9]. For KnGPT2 we apply intermediate layer distillation as well as Vanilla KD. For DistilGPT since the number of layers between the teacher and the student are different, it is not clear which teacher layer should be distilled to which student layer. Although there has been work on intermediate distillation for mismatched layers for BERT [16], extensive experimentation is required to conclude the best practice for GPT.

On the dev set results (Table 5), we observe that KnGPT2 performs better than DistilGPT2 for most datasets and on average. If we apply KD we observe that it is better on all datasets compared to DistilGPT2. Another interesting result is that Vanilla KD does not improve DistilGPT2 fine-tuning. The test set results on Table 6 follow the same trend as the dev results. Interestingly KnGPT2 with KD reaches close to the GPT-2_{Small} performance on average.

[4] This number is presented in [20] for training DistilBERT by the same authors. That uses the same KD algorithm and dataset for pre-training but is applied to BERT rather than GPT. Using a similar hardware we expect this number to be larger for DistilGPT

Model	CoLA	RTE	MRPC	SST-2	MNLI	QNLI	QQP	Average
GPT-2$_{\text{Small}}$	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 5: This table shows performance of the models on dev set of GLUE tasks. Note that GPT-2$_{\text{Small}}$ is used as teacher for KD.

Model	CoLA	RTE	MRPC	SST-2	MNLI	QNLI	QQP	Average
GPT-2$_{\text{Small}}$	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 6: This table shows performance of the models on test set of GLUE tasks. Note that GPT-2$_{\text{Small}}$ is used as teacher for KD.

4.4 Ablation Study

We performed an experiment to study the effect of KD on the pre-training of KnGPT2. In this experiment we used Wikitext-103 as our pre-training dataset. We compare four models and evaluate on LM as well as on classification using the MNLI dataset from GLUE. As shown on Table 7 we compare KnGPT2 without pre-training, with language modeling pre-training only, with KD pre-training only and both language modeling and KD pre-training. Note that we apply ILKD, discussed before for fine-tuning, as our KD algorithm. We observe that pre-training is important for good performance on the downstream task but lower perplexity on LM is not always a good indicator of better downstream performance.

Model	Wikitext-103(pp.)	MNLI (f1)
KnGPT2	28608	69.33
KnGPT2 + LM	**21.94**	77.87
KnGPT2 + KD	144.1	77.50
KnGPT2 + LM + KD	23.04	**77.97**

Table 7: Ablation on the effect of pre-training with KD on language model and MNLI classification

5 Conclusion

In this paper, we compressed GPT-2 by compressing linear layers of a GPT model using Kronecker decomposition. Our model is pre-trained on a relatively small (10 times smaller than the dataset used for baseline) dataset which makes the pre-training much faster. Our proposed model significantly outperformed the baseline on the GLUE benchmark. Using KD can help to further reduce the performance drop of the compressed model. Using Kronecker decomposition on larger GPT models and for higher compression factors are two interesting future directions.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] A. Gokaslan and V. Cohen. Openwebtext corpus, 2019. `http://Skylion007.github.io/OpenWebTextCorpus`.

[4] H. Goldstein. *Multilevel statistical models*, volume 922. John Wiley & Sons, 2011.

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

[6] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[7] H. V. Henderson, F. Pukelsheim, and S. R. Searle. On the history of the kronecker product. *Linear and Multilinear Algebra*, 14(2):113–120, 1983.

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

[9] 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.

[10] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[11] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: an approach to modeling networks. *Journal of Machine Learning Research*, 11(2), 2010.

[12] 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.

[13] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021.

[14] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[15] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models.

[16] 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 Intelligence, 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.

[17] G. Prato, E. Charlaix, and M. Rezagholizadeh. Fully quantized transformer for machine translation. *arXiv preprint arXiv:1910.10485*, 2019.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[19] A. Rashid, V. Lioutas, and M. Rezagholizadeh. Mate-kd: Masked adversarial text, a companion to knowledge distillation. *arXiv preprint arXiv:2105.05912*, 2021.

[20] 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.

[21] 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.

[22] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

[23] M. S. Tahaei, E. Charlaix, V. P. Nia, A. Ghodsi, and M. Rezagholizadeh. Kroneckerbert: Learning kronecker decomposition for pre-trained language models via knowledge distillation, 2021.

[24] 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.

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

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

[27] 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. 2019. In the Proceedings of ICLR.

[28] 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. *arXiv preprint arXiv:2002.10957*, 2020.

[29] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

[30] X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[31] W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812*, 2020.

[32] S. Zhou and J.-N. Wu. Compression of fully-connected layer in neural network by kronecker product. *arXiv preprint arXiv:1507.05775*, 2015.