Compressing Pre-trained Language Models using Progressive Low Rank Decomposition

Habib Hajimolahoseini[*], **Mehdi Rezagholizadeh**[†], **Vahid Partovinia**[†], **Marzieh Tahaei**[†]
Omar Mohamed Awad[*], **Yang Liu**[*]
[*]Ascend Team, Toronto Research Center, Huawei Technologies
[†]Huawei Noah's Ark Lab
habib.hajimolahoseini@huawei.com

Abstract

In this paper, a progressive low rank decomposition method is used to compress large-scale pre-trained transformer based language models. To this end, each fully-connected layers of the transformer modules are decomposed into two consecutive smaller ones using a progressive Singular Value Decomposition technique. In contrast to many of state-of-the-art compression methods where intensive pre-training of the compressed model is necessary, progressive LRD can provide promising performance by compressing the model in the fine-tuning stage. Furthermore, the current state-of-the-art model compression techniques usually face a limitation in their compression ratio as the accuracy gap becomes significant with compression ratios higher than $2\times$. We show that in later steps of the iterative compression where the decomposed models becomes much smaller than their original (compression factors larger than $8\times$), Knowledge Distillation can also be used to improve the performance.

1 Introduction

Over the past few years, Deep Neural Networks (DNNs) have become larger in terms of their number of parameters to gain more performance. In some of the state-of-the-art DNN architecturessuch as Transformer-based pre-trained language models (PLMs), the number of parameters can be in the order of billions (Radford et al., 2018). Training such huge models is a very expensive process, which also requires huge amount of memory (Dean et al., 2012). With a widespread application of real-time AI models on smartphones and other embedding devices, deploying these over-sized models on device can be a very interesting yet challenging target. Compressing deep neural models is a well-known potential solution with a set of diverse existing approaches to improve the efficiency of DNNs at the cost of compromising their performance.

Current model compression algorithms may use different techniques for reducing the number of parameters or computational complexity including: Low Rank Decomposition (LRD), pruning, quantization and Knowledge Distillation (KD) (Cheng et al., 2017). In quantization-based techniques, model weights are approximated using a smaller number of bits e.g. 16-bits, 8-bits or even a single bit (binary networks) (Wu et al., 2016; Han et al., 2015; Courbariaux et al., 2015; Prato et al., 2019; Bie et al., 2019). Model pruning is another approach which removes non-important neurons or redundant connections between them (Lebedev and Lempitsky, 2016; Wen et al., 2016). Although model pruning is shown to be effective for reducing the memory consumption, it is usually not helpful in increasing the inference or training speed significantly (Cheng et al., 2017). Both of these methods try to remove redundancies from the models which in turn helps improving the model generalization too (Gong et al., 2014).

On the other hand, KD approaches transfer the knowledge from a larger model (teacher) into a smaller one (student) by adding and auxiliary loss to imitate softmax outputs or logits from the teacher as a representation of class distributions (Hinton et al., 2015). The goal is for the student model to mimic the behaviour of the teacher model and the knowledge could be distilled from the teachers output or from the intermediate layers (Mirzadeh et al., 2020). The rational behind how KD works is still an open question in the literature, however, there are some works explaining the contribution of adding class similarity information in the output of the teacher (which is reffered to as the *dark knowledge* as well), or regularization effects of the KD loss as potential reasons. Moreover, there are different variants of KD such as intermediate layer distillation, data augmentation for KD, and gradual training to improve its generalizability in different applications. In KD, the student model is usually a shallower and/or narrower version of the teacher. This is usually achieved by layer truncation, in which some of the intermediate layers of the model is removed using a layer selection scheme (Rashid et al., 2021). However, layer truncation methods have no mathematical support and face some serious limitations in high compression ratios (ratios more than $2\times$) because truncating a high portion of the layers may lead to a significant accuracy drop (Cheng et al., 2017).

In contrast, LRD techniques decompose each layer of DNNs using a tensor or matrix factorization method. LRD can be applied to both fully-connected and convolutional layers, using a matrix or tensor factorization technique, respectively. One advantage of LRD methods is that they do not need heavy pre-training because they start from a large pre-trained model to initialize the compressed model. This is in contrast to the layer truncation technique where we should train the compressed model from scratch. Therefore when using LRD, a few fine-tuning steps are enough to recover most of the accuracy drop. However, it is worth mentioning that when using LRD for large compression factors, the approximation of the original weights of the compressed model becomes very poor and thus recovering the accuracy through fine-tuning becomes very difficult. Progressive LRD (Gusak et al., 2019b) is a way of addressing this issue. In progressive LRD, compression is applied in small steps iteratively, therefore in each step, the compressed model is an approximation of the previous model and thus has a better initialization point on the loss surface (Gusak et al., 2019b).

Although prior arts (Mao et al., 2020) have used SVD for compression of large PLMs, this technique has only be used in a one stage setup. In this paper, we apply a progressive low rank decomposition method to investigate its performance on the extreme compression of Transformer-based PLMs in NLP. In this regard, we first decompose each linear mapping in Transformers into two smaller consecutive ones using an SVD decomposition method. The decomposition rank is estimated based on the desired compression ratio for each layer. We estimate the ranks in a progressive way in which the entire model is first decomposed altogether with a large rank (small compression) and then fine-tuned for some epochs to recover the accuracy. Another round of compression is applied again with a smaller rank and the previous step is repeated until the desired compression ratio is achieved.

2 Low Rank Decomposition

The transformer modules in the NLP models include feed forward and multi-head attention modules, in which the trainable blocks are fully-connected (FC) layers (Vaswani et al., 2017). The FC layers have a 2D weight matrix W shaped as a two dimensional tensor of size (C, S), where C and S represent the number of input and output features, respectively. Suppose that the input to the FC layer is a two dimensional tensor $\mathbf{X}$ of size (B, C), where B and C are the batch size and the number of input features, respectively. The FC layer will multiply the input tensor $\mathbf{X}$ with its weight matrix and deliver the output $\mathbf{Y}$ of shape (B, S) through $\mathbf{Y} = \mathbf{XW}$. In most NLP models, the dimensions of FC layers could be so large that their weight matrix includes millions of parameters only in a single FC layer.

For example, the shape of a FC layer in the GPT2 model is $(10240, 2560)$, surpassing 26 million parameters. Mathematically speaking, the number of parameters in the FC layer weight $\mathbf{W}$ is the multiplication of the tensor size: $N = CS$. However, if we decompose it into two smaller tensors of size (C, R) and (R, S) whose matrix product has the same shape as the original matrix, the number of parameters shrinks significantly: $M = R(C + S)$, where M represent the total number of parameters in the decomposed layer and R is the rank of the decomposition. According to these two equations, the compression ratio P is defined as

$$P = \frac{N}{M} = \frac{CS}{R(C + S)} \tag{1}$$

The rank of a matrix W can be interpreted in different ways, as the number of its linearly independent columns or rows sometimes even the condition index, condition number, normalized trace or determinants are used as proxy to the rank. The low rank approximation of matrix $\mathbf{W}$ is an optimization problem which minimizes the difference between $\mathbf{W}$ and its low rank version. In terms of Singular Value Decomposition (SVD), this optimization problem has an analytical solution as follows (Van Loan, 1987): Assume that the weight matrix $\mathbf{W} \in \mathrm{I\!R}^{\mathrm{C}\times\mathrm{S}}$ is decomposed using SVD as follows

$$W = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top, \tag{2}$$

where $\mathbf{U} \in \mathrm{I\!R}^{\mathrm{C}\times\mathrm{C}}$ and $\mathbf{V} \in \mathrm{I\!R}^{\mathrm{S}\times\mathrm{S}}$ are the orthogonal matrices and $\mathbf{\Sigma} \in \mathrm{I\!R}^{\mathrm{C}\times\mathrm{S}}$ is a diagonal rectangular matrix containing the singular values $\sigma_i >$ of $\mathbf{W}$ of rank r. In (2), if we only use the first R terms of the summation, the resulted matrix $\mathbf{W}'$ would be an approximation of $\mathbf{W}$ with a lower rank $R < r$:

$$\mathbf{W}' = \sum_{i=1}^{R} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = \mathbf{U}'\mathbf{\Sigma}'\mathbf{V}'^\top \tag{3}$$

where $\mathbf{U}' \in \mathrm{I\!R}^{\mathrm{C}\times\mathrm{R}}$ and $\mathbf{V}' \in \mathrm{I\!R}^{\mathrm{S}\times\mathrm{R}}$ are the new orthogonal matrices and $\mathbf{\Sigma}' \in \mathrm{I\!R}^{\mathrm{R}\times\mathrm{R}}$ is the new diagonal rectangular matrix. Therefore, the low rank estimation problem could be explained as the following minimazation

$$\min_{\mathrm{rank}(\mathbf{X})\le R} \|\mathbf{W}-\mathbf{X}\|_2 = \|\mathbf{W}-\mathbf{W}'\|_2 . \tag{4}$$

According to (3), a QR-type reconstruction of $\mathbf{W}'$ is possible multiplication the following two matrices

$$\mathbf{W}' = \mathbf{W}_0\mathbf{W}_1, \quad \mathbf{W}_0 = \mathbf{U}'\sqrt{\mathbf{\Sigma}'}, \quad \mathbf{W}_1 = \sqrt{\mathbf{\Sigma}'}\mathbf{V}'^\top \tag{5}$$

where $\mathbf{W}_0 \in \mathrm{I\!R}^{\mathrm{C}\times\mathrm{R}}$ and $\mathbf{W}_1 \in \mathrm{I\!R}^{\mathrm{R}\times\mathrm{S}}$, and $\sqrt{\mathbf{\Sigma}}$ is a diagonal of square root of singular values $\sqrt{\sigma_i}$. Based on 1, it is clear that if we replace the FC layer $\mathbf{W}$ with two consecutive FC layers $\mathbf{W}_0$ and $\mathbf{W}_1$, the number of parameters could shrink significantly depending on the rank R.

Different approaches are proposed to estimating the rank R in practice. A simple approach is to specify the rank based on the desired compression ratio for each FC layer. Thus, using 1 if we want the FC layer to be compressed by a factor of P, the rank R is $R = \frac{CS}{P(C+S)}$. The range of the rank R is quite important. If R is large, the product of decomposed matrices $\mathbf{W}_0$ and $\mathbf{W}_1$ would be closer to the original matrix $\mathbf{W}$ and the drop in accuracy is negligible with the price of small compression or even no compression. The square $\mathbf{\Sigma}$of size $R \times R$ fits within the rectangle $\mathbf{W}$ of size $C \times S$, if $R \le \min(C, S)$, which is a standard result in linear algebra.

On the other hand, if R is too small, the approximation in 3 is inaccurate and accuracy loss is significant with the advantage of high compression ratio. In the extreme case of $R = 1$, the matrix $\mathbf{\Sigma}'$ becomes an scalar and thus, $\mathbf{U}'$ and $\mathbf{V}'$ reduce to vectors. Therefore, the upper and lower bounds of the rank R and their respective compression ratio P is

$$\frac{CS}{\min(C,S)(C+S)} \le P \le \frac{CS}{C+S}. \tag{6}$$

Define a hidden node of a single FC layer as

$$f_{\mathbf{W}}(x) = \mathbf{w}_2^\top a\left(\mathbf{W}\mathbf{x}\right),$$

which is used to approximate a continuous function $f(\mathbf{x})$ on a compacta $K \subset \mathrm{I\!R}^{\mathrm{C}}$, in which $a(\cdot)$ is an activation function. The following theorem shows under some bound on singular values, a low rank FC layer behaves similar to a full rank layer.

Theorem 1 *A FC layer of rank R and n hidden units defined on a compacta $K \subset \mathrm{I\!R}^{\mathrm{C}}$ with L-Lipschitz activation is can be approximated any continuous $f \in C(K)$ with error $(\epsilon + \delta)$, for a large enough n given*

$$\sigma_{r+1} < \delta L^{-\frac{1}{2}} \left(\|K\|_2 \|\mathbf{w}_2\|_2\right)^{-1}$$

where σ_{r+1} is the $r+1$ singular value of the original weight matrix $\mathbf{W}$, ϵ is the approximation quality of the full rank neural network $\|f(\mathbf{x}) - f_{\mathbf{W}}(\mathbf{x})\|_2 < \epsilon$, and δ is the approximation error added due to the low rank decomposition.

See Appendix for the proof.

3 Progressive Decomposition

We can use different strategies for applying the low rank decomposition to a deep model. It could be applied to all layers of the model at the same time, or it could be done layer by layer. However, due to the aggressive target compression ratio, we decompose the NLP models in multiple intermediate steps (applying SVD to all layers in case of LRD) instead of one-shot compression, where we follow each compression step with some finetuning epochs. This method is actually inspired by the method proposed in (Gusak et al., 2019a) for the convolutional networks. To this end, we use a progressive rank selection approach in which, we start from large values for R and apply low rank decomposition to the entire model altogether with a low compression ratio.

After applying each LRD stage, the accuracy will drop. Therefore, the model is fine-tuned for few epochs in order to recover the accuracy. The rank R is decreased in each stage so that we have a higher compression ratio. Note that the number of layers will not be doubled again in the 2nd stage of decomposition as we use matrix multiplication for merging the newly generated weight matrices in order to keep the number of layers the same as the 1st stage of LRD (Gusak et al., 2019a). This process is shown in Fig. 1.

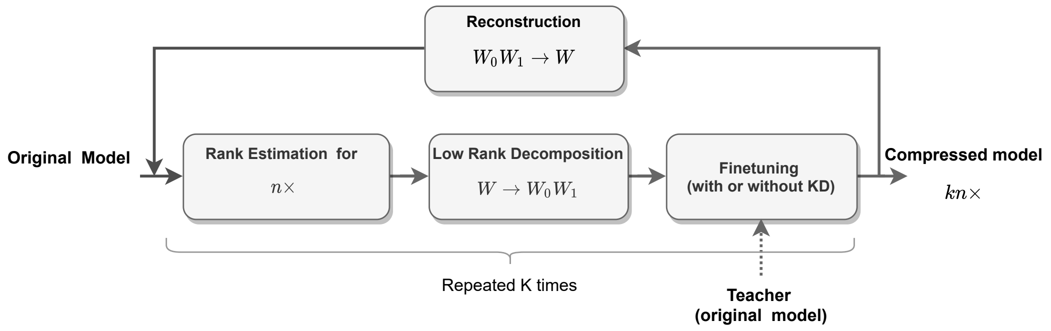


Figure 1: Progressive decomposition scheme

4 Knowledge Distillation

During the fine-tuning of the decomposed model, we also apply a knowledge distillation technique which transforms knowledge from the original fine-tuned model (teacher) to the decomposed model (student) in order to help recovering the accuracy of the decomposed model even faster (Hinton et al., 2015). More specifically, we use the Mate-KD technique proposed in (Rashid et al., 2021) as our KD methodology. In Mate-KD an adversarial text generator is trained alongside the student network. The generator is trained to perturb samples to maximize the divergence between the student's and the teacher's logits. The student is trained to minimize the KD loss on the original and the perturbed samples. Through this adversarial training the performance of KD can be improved compared to other state-of-the-art approaches (Rashid et al., 2021).

5 Experimental Results

In order to evaluate the LRD approach, we use the well-known pre-trained language model GPT2 as our baseline and apply different compression techniques to it (Radford et al., 2019). The baseline GPT2 model includes 12 Transformer layers. In our experiments, we also include three truncated versions of GPT2 including DistilGPT2, GPT2-6L and GPT2-4L which are the truncated models with 6, 6 and 4 Transformer layers, respectively. DistilGPT2 is in fact the official compressed version of GPT2 which includes 6 layers of Transformer layers and is trained under supervision of GPT2 as its teacher using knowledge distillation. The GPT2-6L is a similar truncated model which includes only the Transformer layers 0, 2, 4, 7, 9 and 11 of the baseline GPT2. In GPT2-4L, only the layers 0, 4, 7 and 11 are used. We use the standard Wikitext-103 benchmark dataset (Merity et al., 2016) in which the perplexity is the measure of performance.

On the other hand, we also apply two versions of LRD method, one with a single step decomposition another one thet applies 2 rounds of $2\times$ compression to the FC layers inside each Transformer module. In GPT2, there are 4 FC layers inside each Transformer unit. The shapes of the FC layers and the ranks after each round of $2\times$ compression is shown in Table 1. The comparison results for all the methods are also reported in Table 2.

Table 1: The shapes of FC layers inside each Transformer block in GPT2 model and the calculated ranks after each compression step.

Name of the FC Layer	**Input**	**Output**	**Rank** $2\times$	**Rank** $4\times$
query key value	768	2304	288	144
dense	768	768	192	96
dense h to 4h	768	3072	306	153
dense 4h to h	3072	768	306	153

Table 2: Experimental results for GPT2 using low rank decomposition v.s. layer truncation on Wikitext-103 dataset. GPT2-6L and GPT2-4L are the truncated versions of GPT2 with 6 and 4 Transformer layers, respectively. The LRD result is after applying 2 rounds of $2\times$ compression (total of $4\times$).

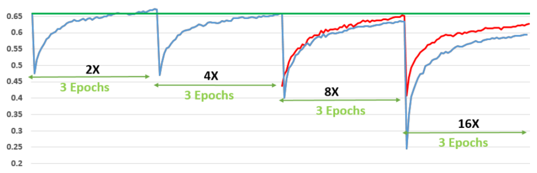
Method	**# Layers**	**# Params (M)**	**Perplexity**
GPT2 (Baseline)	12	124	16.3
Distiltgpt2	6	82	21.1
GPT2-6L	6	82	22.7
GPT2-4L	4	68	30.0
Single Step LRD	12	47	28.2
Progressive LRD	12	31	22.0

As seen in Table 2, the progressive LRD model is able to reach a perplexity close to DistilGPT2, with only 0.9 unit gap in perplexity, while providing a much higher compression ratio of $4\times$ comparing to that of truncated versions e.g. DistilGPT2 which is $1.5\times$. The results for GPT2-4L also shows that if we truncate 2 more layers of the GPT2 to get a higher compression ratio, we will start to lose the perplexity significantly (around 9 perplexity units) while the compression ratio will only increase from $1.5\times$ to $1.84\times$. Note that in our LRD experiments here, we also decompose the last fully connected layer which maps the features from the last Transformer layer to the number of classes at the output. This fully connected layer has a dimension of 768 by 50257 for the Wikitext-103 experiment. In the layer truncation method, this layer can not be removed however.

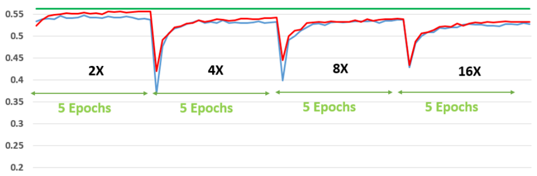
In order to evaluate the progressive LRD on a different language e.g. Chinese, we use a large scale generative pre-trained language model called CPM (Zhang et al., 2021). CPM is in fact the largest Chinese pretrained language model and could be considered as a Chinese version of its English counterpart GPT (Radford et al., 2019). Simillar to the GPT, the CPM model comes in different versions according to the number of Transformer layers it consists of. More specifically, we use the CPM-Small with 12 and CPM-Large with 32 Transformer layers who have 109 million and 2.6 billion parameters, respectively.

We use three Chinese datasets in our experiments called CHID (Zheng et al., 2019), STC (Shang et al., 2015) and TNEWS (Xu et al., 2020). For the CPM-Small, we compress the FC layers inside each Transformer module up to 16 times progressively with $2\times$ compression for 4 times. This will compress the entire CPM-Small model up to $3.8\times$ compression. This process is shown in Fig.2.

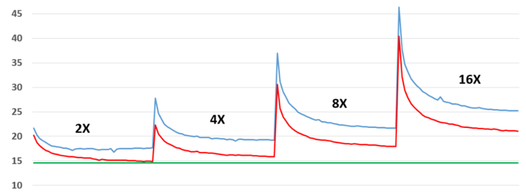
As seen in Fig.2a, for the CHID dataset, applying $2\times$ compression to the CPM-Small will result in even a higher accuracy after 3 epochs of fine-tuning. This is because in the new compressed model, the redundancy is reduced and therefore the $2\times$ decomposed model is more robust to over-fitting. After the second round of $2\times$ compression to the FC layers, the accuracy is also fully recovered after 3 epochs of fine-tuning. Therefore, we didn't apply any Mate-KD up to this point as there is no accuracy gap between the original and LRD models. As we apply the third round of $2\times$ decomposition to the CPM-Small model, the accuracy could not be fully recovered after 3 epochs. Therefore, we apply the Mate-KD technique and repeat the fine-tuning (the red curve in Fig.2a). As seen in this figure, the accuracy could again be fully recovered with the help of Mate-KD. However, after applying the forth round of $2\times$ compression, the decomposed model shows an accuracy drop of



(a) Validation accuracy of CPM-Small with LRD on CHID dataset



(b) Validation accuracy of CPM-small with LRD on Tnews dataset



(c) Validation Perplexity of CPM-small with LRD on STC dataset

Figure 2: Accuracy during progressive LRD of CPM-small (per evaluation step. The green line shows the best accuracy of the original model. The red and blue curves show the LRD finetuning with and without Mate-KD, respectively.)

6.66% after 3 epochs of fine-tuning comparing to the original model. After applying the Mate-KD, this accuracy gap is reduced to 3.3%.

On the other hand, for the CPM-Large, we first start with a $3\times$ compression and then aply 2 times of $2\times$ compression steps. The reason for applying $3\times$ compression at the first step is that the CPM-Large model is so big that even after the first $2\times$ compression the model itself could not be fit on a single GPU. The total compression of FC layers would finally be $12\times$ which results of $9\times$ compression of the entire CPM-Large model. The summary of experimental results for the CPM-Small and CPM-Large on the benchmark dataset is reported in Table 3. The experimental results for CPM-Small with LRD is reported with and without Mate-KD.

Table 3: Experimental results for CPM-Small and CPM-Large using progressive low rank decomposition with $3.7\times$ and $9\times$ compressions, respectively.

Data Measure	# Params Millions	CHID Accuracy	Tnews Accuracy	STC Perplexity
CPM-Small	109	66.0	56.2	14.6
LRD	29	59.3	53.0	25.2
LRD+Mate-KD	29	62.7	53.4	21.1
CPM-Large	2600	80.8	59.9	-
LRD	288	72.3	54.0	-

6 Conclusion

In this work, a progressive low rank decomposition method was used for compression of large transformer based language models. In contrast to many of state-of-the-art compression methods where intensive pre-training of the compressed model is necessary, progressive LRD can provide promising performance by compressing the model in the fine-tuning stage. This leads to reduction in the computation resources needed for obtaining a compressed model for a given task. We show that in later steps of the iterative compression where the student models becomes much smaller than the teacher (compression factor larger than $8\times$) KD can be used to improve the performance.

References

Alex Bie, Bharat Venkitesh, Joao Monteiro, Md Haidar, Mehdi Rezagholizadeh, et al. 2019. A simplified fully quantized transformer for end-to-end speech recognition. arXiv preprint arXiv:1911.03604.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In Advances in neural information processing systems, pages 3123–3131.

Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al. 2012. Large scale distributed deep networks.

Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. Psychometrika, 1(3):211–218.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115.

Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. 2019a. Automated multi-stage compression of neural networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops, pages 0–0.

Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Ivan Oseledets, and Andrzej Cichocki. 2019b. Musco: Multi-stage compression of neural networks. arXiv preprint arXiv:1903.09973.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251–257.

Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2554–2564.

Yihuan Mao, Yujing Wang, Chufan Wu, Chen Zhang, Yang Wang, Yaming Yang, Quanlu Zhang, Yunhai Tong, and Jing Bai. 2020. Ladabert: Lightweight adaptation of bert through hybrid model compression. arXiv preprint arXiv:2004.04124.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. arXiv preprint arXiv:1609.07843.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2020. Improved knowledge distillation via teacher assistant. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 5191–5198.

Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. 2019. Fully quantized transformer for machine translation. arXiv preprint arXiv:1910.10485.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9.

Ahmad Rashid, Vasileios Lioutas, and Mehdi Rezagholizadeh. 2021. Mate-kd: Masked adversarial text, a companion to knowledge distillation. arXiv preprint arXiv:2105.05912.

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. arXiv preprint arXiv:1503.02364.

Charles Van Loan. 1987. Matrix computations and signal processing. Technical report, Cornell University.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. Advances in neural information processing systems, 29:2074–2082.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized convolutional neural networks for mobile devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4820–4828.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, et al. 2020. Clue: A chinese language understanding evaluation benchmark. arXiv preprint arXiv:2004.05986.

Zhengyan Zhang, Xu Han, Hao Zhou, Pei Ke, Yuxian Gu, Deming Ye, Yujia Qin, Yusheng Su, Haozhe Ji, Jian Guan, et al. 2021. Cpm: A large-scale generative chinese pre-trained language model. AI Open, 2:93–99.

Chujie Zheng, Minlie Huang, and Aixin Sun. 2019. Chid: A large-scale chinese idiom dataset for cloze test. arXiv preprint arXiv:1906.01265.

Appendix

Proof of Theorem 1 *We drop the bias term for the simplicity of notation. We need to bound*

$$\begin{aligned}\|f - f_{\mathbf{W}_R}\|_2^2 &= \int_K \{f(\mathbf{x}) - f_{\mathbf{W}_R}(\mathbf{x})\}^2 d\mu \\ &= \int_K \{f(\mathbf{x}) - f_{\mathbf{W}}(\mathbf{x})\}^2 d\mu \qquad (7)\\ &+ \int_K \{f_{\mathbf{W}}(\mathbf{x}) - f_{\mathbf{W}_R}(\mathbf{x})\}^2 d\mu \qquad (8)\\ &+ 2\int_K \{f(\mathbf{x}) - \hat{f}_{\mathbf{W}}(\mathbf{x})\}\{f_{\mathbf{W}}(\mathbf{x}) - f_{\mathbf{W}_R}(\mathbf{x})\} d\mu \qquad (9)\end{aligned}$$

The full rank version $f_{\mathbf{W}}$ is dense in $C(K)$ according to (Hornik, 1991), therefore (7) *is bounded by ϵ^2. It is only required to show that* (8) *is also bounded*

$$\begin{aligned}&\int_K \{f_{\mathbf{W}}(\mathbf{x}) - f_{\mathbf{W}_R}(\mathbf{x})\}^2 d\mu \\ =& \int_K \left\{\mathbf{w}_2^\top a\left(\mathbf{W}\mathbf{x}\right) - \mathbf{w}_2^\top a\left(\mathbf{W}_R\mathbf{x}\right)\right\}^2 d\mu \\ \leq& \; L\,\|\mathbf{w}_2\|_2^2\,\|K\|_2^2\,\sigma_{r+1}^2,\end{aligned}$$

where the last inequality is low rank spectral norm bound (Eckart and Young, 1936) . The last term (9) *is consequently bounded by the Cauchy-Schwarz inequality as follows*

$$\begin{aligned}\left[\int_K \{f(\mathbf{x}) - \hat{f}_{\mathbf{W}}(\mathbf{x})\}\{f_{\mathbf{W}}(\mathbf{x}) - f_{\mathbf{W}_R}(\mathbf{x})\} d\mu\right]^2 &\leq \int_K \{f(\mathbf{x}) - \hat{f}_{\mathbf{W}}(\mathbf{x})\}^2 d\mu \int_K \{f_{\mathbf{W}}(\mathbf{x}) - f_{\mathbf{W}_R}(\mathbf{x})\}^2 d\mu \\ &\leq \epsilon^2 L\,\|\mathbf{w}_2\|_2^2\,\|K\|_2^2\,\sigma_{r+1}^2.\end{aligned}$$

Therefore,

$$\|f - f_{\mathbf{W}_R}\|_2^2 \;\leq\; (\epsilon + \sqrt{L}\,\|\mathbf{w}_2\|_2\,\|K\|_2\,\sigma_{r+1})^2,$$

consequently bounding the last term with $(\epsilon+\delta)^2$ requires

$$\sigma_{r+1} \leq \delta L^{-\frac{1}{2}}(\|\mathbf{w}_2\|_2\,\|K\|_2)^{-1}$$