RETHINKING COMPRESSION: REDUCED ORDER MODELLING OF LATENT FEATURES IN LARGE LANGUAGE MODELS

Arnav Chavan[*1,2], **Nahush Lele**[*1], **Deepak Gupta**[2]
[1]Nyun AI [2]Transmute AI Lab (Texmin Hub), IIT (ISM) Dhanbad, India
arnav.chavan@nyunai.com, guptadeepak2806@gmail.com

ABSTRACT

Due to the substantial scale of Large Language Models (LLMs), the direct application of conventional compression methodologies proves impractical. The computational demands associated with even minimal gradient updates present challenges, particularly on consumer-grade hardware. This paper introduces an innovative approach for the parametric and practical compression of LLMs based on reduced order modelling, which entails low-rank decomposition within the feature space and re-parameterization in the weight space. Notably, this compression technique operates in a layer-wise manner, obviating the need for a GPU device and enabling the compression of billion-scale models within stringent constraints of both memory and time. Our method represents a significant advancement in model compression by leveraging matrix decomposition, demonstrating superior efficacy compared to the prevailing state-of-the-art structured pruning method.

1 INTRODUCTION

Recent advances in generative modeling have led to a notable increase in the construction of large language models (LLMs), some of which consist of hundreds of billions of parameters. Despite their commendable accuracy, the associated computational demands are considerable, particularly in terms of GPU memory for inference. In practical applications, there is a growing need to compress these models while minimizing the accompanying performance degradation.

Promising approaches to compress LLMs include pruning (Frantar & Alistarh, 2023; Sun et al., 2023; Ma et al., 2023), quantization (Frantar et al., 2022; Dettmers et al., 2023) and knowledge distillation (Wu et al., 2023; Gu et al., 2023). Current LLM quantization methods require specific hardware-level support and are not able to reduce MACs and speed up inference time due to expensive quant-dequant operations in LLMs. Knowledge distillation has been shown to perform well in a training-aware fashion on standard deep learning models. However, the massive computational resources needed for distillation limits the applicability of such approaches. Recently, (Ma et al., 2023) presented a structured pruning approach designed for LLMs. While this approach is capable of pruning the LLMs with no need for fine-tuning, the drop in performance is significant, and clearly there is a need to explore further in this direction. Moreover, the pruning strategy is not universal and significant effort is needed per neural architecture to identify the prunable structures.

In this paper, we present a novel, practical and training-free approach to model compression which is specifically for large models including LLMs. Referred further as LLM-ROM, our approach performs localized reduced order modelling of the latent features through low-rank decomposition in the feature space and re-parameterization in the weight space. Since LLM-ROM operates layerwise, it does not require any massive model updates and can be executed on small GPU/CPU resources. The simplicity of LLM-ROM facilitates the compression of billion-scale models within stringent constraints of both memory and time. Our early experiments demonstrate that LLM-ROM outperforms existing approaches and can compress LLMs without any fine-tuning.

[*]Equal contribution. Work done while Nahush was an Intern at Nyun AI

2 METHOD

LLM-ROM builds reduced-order model (ROM) layerwise, and for a model with L layers, the decomposition of the latent feature maps is done in a sequential manner using a calibration data $\mathbf{X} \in \mathbb{R}^{B \times d_1}$, where B and d_1 denote the batch-size and number of input channels, respectively. For the i^{th} layer, denoted as L_i with weights $\mathbf{W}_i \in \mathbb{R}^{d_2 \times d_1}$ where d_2 denotes the output channels, we compute the feature map $\mathbf{Y}_i = \mathbf{W}_i \mathbf{X}_i \in \mathbb{R}^{B \times d_2}$. Following this, the principal components of $\mathbf{Y}_i$ are then computed through eigenvalue decomposition of the symmetric covariance matrix of Y_i. These components can be represented as $V_j \in \mathbb{R}^{d_2} \quad \forall \; j \in [1, d_2]$, and the principal component matrix can be represented as $\mathbf{V} \in \mathbb{R}^{d_2 \times d_2}$, with each row denoting a principal component arranged in the descending order of their eigenvalue.

Depending upon the target rank of the layer, we select only the top r principal components ranked by their respective eigenvalues. Thus, we index $V_r = \mathbf{V}[1 \rightarrow r, :] \in \mathbb{R}^{r \times d_2}$. Thus, the ROM of this layer can be denoted as $\mathbf{Y}_i = \mathbf{V}_r^T \mathbf{V}_r \mathbf{W}_i \mathbf{X}_i$. Upon re-parameterization into low-rank matrices, $\mathbf{W}_{i1} = \mathbf{V}_r^T \in \mathbb{R}^{d_2 \times r}$ and $\mathbf{W}_{i2} = \mathbf{V}_r \mathbf{W}_i \in \mathbb{R}^{r \times d_1}$, the layer can be decomposed into a sequential combination of two smaller linear layers with weights $\mathbf{W}_{i1}$ and $\mathbf{W}_{i2}$ respectively. We consider the ROM of the previous layer to generate inputs for the next layer so that the next layers have prior information of the error introduced in the previous layers for decomposition. Note that the ROM operations are performed on CPU with no requirement for a GPU, and the computational cost associated with it is very small. Details are presented in Appendix C.

3 EXPERIMENTS

To evaluate the performance of the model in a task-agnostic setting, we employ LLaMA's (Touvron et al., 2023) assessment methodology, performing zero shot task classification across common sense reasoning datasets, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy (Clark et al., 2018), and ARC-challenge (Clark et al., 2018). We use a batch-size of 512 for the calibration data (Appendix A.3) from the training splits of the aforementioned datasets, ensuring no data leakage, and set a maximum sequence length of 128. We adopt uniform compression rate across layers during the decomposition process, commencing from a specific layer and bypassing the preceding layers to achieve the overall target compression. Further details on the estimation of layerwise ranks can be found in the Appendix. We set target compression rates of 80%, and 50% and compare with LLM-Pruner[1] with and without fine-tuning in Table 1.

Our LLM-ROM method consistently outperforms LLM-Pruner at 80% and 50% compression without any fine-tuning. It is noteworthy that at 80% budget our method even outperforms fine-tuned LLM-Pruner model, signifying that ROM is able to better extract smaller neural structures and weights from larger counterparts without any gradient updates on the extracted weights.

Method	Finetune	#Params	#MACs	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	Average
LLaMA-7B	-	6.7B	423.93G	76.5	79.8	76.1	70.1	72.8	47.6	70.5
LLM-Pruner	✗	5.4B	339.60G	59.4	75.6	65.3	61.3	59.2	37.1	59.7
LLM-Pruner	✓	5.4B	339.60G	69.5	76.4	68.1	65.1	63.4	37.9	63.4
LLM-ROM	✗	5.4B	339.99G	74.5	73.8	66.6	68.1	67.2	39.8	65.0
LLM-Pruner	✗	3.4B	206.59G	52.3	59.6	35.6	53.2	33.5	27.2	43.6
LLM-Pruner	✓	3.4B	206.59G	60.3	69.3	47.1	53.4	46.0	29.2	50.9
LLM-ROM	✗	3.5B	215.61G	62.0	62.5	35.3	57.7	39.3	27.6	47.4

Table 1: Comprehensive comparison of our method with LLM-Pruner on LLaMA-7B model.

4 CONCLUSION

In this paper we presented a new direction for LLM compression leveraging reduced order modeling of the latent features. Based on the concept of identifying the finite set of most useful latent feature modes, LLM-ROM is capable of compressing LLMs without the need for any fine-tuning. With no requirement of a GPU during the compression process, LLM-ROM can be efficiently run on a simple CPU machine. Moreover, unlike pruning, LLM-ROM is very generic and does not require manual interference for different model architectures. Based on the presented results, we hope to have paved way for a novel approach to design compressed LLMs in a resource-efficient manner.

[1]We pick the best performing pruning method from LLM-Pruner *i.e.* Block Pruning.

URM STATEMENT

The authors acknowledge that at least one key author of this work meets the URM criteria of the ICLR 2024 Tiny Papers Track.

REFERENCES

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language, 2020.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *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)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

X. Ma, G. Fang, and X. Wang. Llm-pruner: On the structural pruning of large language models. *NeurIPS*, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, aug 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL https://doi.org/10.1145/3474381.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji. Lamini-lm: A diverse herd of distilled models from large-scale instructions. *arXiv preprint arXiv:2304.14402*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, abs/1506.06724, 2015. URL http://arxiv.org/abs/1506.06724.

A HYPERPARAMETERS

A.1 LAYERWISE RANK COMPUTATION

The LLaMA-7B (Touvron et al., 2023) model which we use for our experiments consists of 32 identical decoder modules (these modules comprise >96% of the total model parameters), each of which consists of seven decomposable weight matrices. Our initial studies showcased that setting a uniform budget for all the modules from the very beginning of the model lead to a significant deterioration in model performance, for this reason we restrict the application of our compression process to a subset of modules. Further, decomposition of the layers of a module introduces errors in the outputs of that layer which get compounded as we move forward in the network, to minimize this we only compress modules towards the end of the model. Based on these heuristics we perform experiments compressing varying number of modules from the end depending on the budget that we need to satisfy for the entire model. The specific number of modules to be compressed is determined empirically for each budget. For instance, to achieve an overall budget of 80%, we conducted experiments compressing only the last 8 modules uniformly with a budget of 0.20, the last 12 modules with a budget of 0.46, and the last 16 modules with a budget of 0.60. Our findings indicated that compressing the last 12 modules yielded the most favorable results for 80% budget. Similar experiments for 90% and 50% budget yield the best results when we compress the last 8 modules with a budget of 0.60 and the last 24 modules with a budget of 0.33 respectively, the results of which are listed in Table 1. Each module is originally composed of 4 weight matrices $\mathbf{W}_i \in \mathbb{R}^{4096\times4096}$ from the self attention block and 3 weight matrices $\mathbf{W}_j \in \mathbb{R}^{4096\times11008}$ from the feed-forward network (although one of these is transposed, it does not change the computed rank). At the end of compression each weight matrix in the self-attention block is decomposed into two low rank weight matrices $\mathbf{W}_{i1} \in \mathbb{R}^{4096\times r}$ and $\mathbf{W}_{i2} \in \mathbb{R}^{r\times4096}$, with $r = 1228,\ 954$ and 675 and the weight matrices in the feed-forward network are decomposed into $\mathbf{W}_{j1} \in \mathbb{R}^{4096\times r}$ and $\mathbf{W}_{j2} \in \mathbb{R}^{r\times11008}$ with $r = 1791,\ 1373,$ and 985 for the three module budget settings of $60\%, 46\%$ and 33% corresponding to overall model budget of $90\%, 80\%$ and 50% respectively.

A.2 EFFECT OF BATCH SIZE AND SEQUENCE LENGTH

The eigenvalue decomposition of the covariance matrix and the subsequent selection of the principal components require a computation of the outputs of that layer. The batch used to compute this output is a key factor that can influence the generalizability of the layers obtained after decomposition. Principal components computed on a larger sample size will exhibit closer alignment with those of the true distribution. To corroborate this hypothesis, we conducted experiments along two orthogonal directions: one by solely varying the batch-size, and the other with the variation of sequence length, the results for the same are presented in Table 2 and 3 respectively.

Batch Size	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	Average
512	74.5	73.8	66.6	68.1	67.2	39.8	65.0
128	72.6	72.4	63.2	66.3	61.1	37.7	62.2
32	70.2	68.4	58.7	67.2	55.7	35.7	59.3

Table 2: Effect of batch size on model performance at a sequence length 128.

Seq. Length	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	Average
128	74.5	73.8	66.6	68.1	67.2	39.8	65.0
64	66.6	74.4	65.7	67.6	65.4	40.1	63.3
32	66.2	73.4	65.1	67.8	64.6	39.5	62.7

Table 3: Effect of sequence length on model performance at batch size 512.

From Tables 2 and 3 it is evident that a larger batch is beneficial and results in significantly better model generalization and at the same time longer sequence length also aids in maintaining model performance post compression.

A.3 CHOICE OF CALIBRATION DATASET

Given that the activations of data from the calibration dataset are used to calculate the covariance matrix, which is subsequently utilized for eigendecomposition, it is reasonable to infer that the model's performance is sensitive to the choice of this dataset. This inference is supported by our findings in the conducted studies where we use three different datasets, namely ARC-challenge (Clark et al., 2018), BookCorpus(Zhu et al., 2015) and a combination of all the common sense task prompts i.e each batch contains an equal number of samples from the six common sense reasoning tasks' datasets used for benchmarking namely BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy (Clark et al., 2018), and ARC-challenge (Clark et al., 2018) , as our calibration datasets at a budget of 80% keeping other hyperparameters such as batch-size and sequence length constant. When creating the calibration datasets we choose samples from a data split which is disjoint from the set upon which evaluation is performed, ensuring there is no data leak. The results of these studies are compiled in Table 4.

Dataset	**BoolQ**	**PIQA**	**HellaSwag**	**WinoGrande**	**ARC-e**	**ARC-c**	**Average**
Combination	74.5	73.8	66.6	68.1	67.2	39.8	65.0
ARC-c	64.6	72.5	63.8	67.0	67.8	40.9	62.8
Book Corpus	63.2	73.6	65.6	67.7	63.6	38.0	61.9

Table 4: Comparison of model performance with respect to choice of calibration dataset

The results presented above show the influence of the calibration dataset choice on model performance. It is unsurprising that the dataset, which consists of the combination of all common sense tasks used for benchmarking, exhibits the most favorable relative performance.

B COMPUTATIONAL COST

We conduct ROM of LLaMA-7B (Touvron et al., 2023) on a CPU server with 128 GB RAM and 48-core/96-thread processor. Our current implementation loads the complete model at once; however, it is trivial to perform ROM layerwise and hence can be done in under 10 GB of peak RAM given that only inputs and weights of current layer are loaded and processed into the memory. On an average it takes 13 seconds to perform ROM of each layer of LLaMA-7B (Touvron et al., 2023) which has a total of 224 layers. Overall, it takes 15.8 minutes, 21.8 minutes and 28.9 minutes for 90%, 80% and 50% compression rates respectively.

C LIMITATIONS

One inherent limitation of the proposed method is uniform rank distribution across layers. Not all layers contribute equally to model performance and a mechanism to identify the target rank of each layer for a given model compression budget is needed. Since LLM-ROM uses calibration data for performing reduced order modelling of features, out of domain performance may be impacted. For example models compressed using natural language data may not perform very well on code tasks. However this will require a thorough understanding and experimentation.