Compressing Large Language Models

Emma Lei Hovmand
Dept. of Electrical & Computer Engineering
New York University
ech8407@nyu.edu

Lenox Hsu
Dept. of Electrical & Computer Engineering
New York University
yh4508@nyu.edu

Abstract

With the emergence of Large Language Models (LLMs), providing methods of making such models broadly accessible has followed. In this work we present currently available methods of compression including pruning, the LLM.int8() method and the Low Rank Adaptation (LoRA) method. Additionally, we present our findings from implementing LoRA in the multilayer perceptron (MLP) layer of the Generative Pre-trained Transformer 2 (GPT-2) for retraining. Based on our findings, applying the LoRA method technique in the MLP layer of the GPT-2 LLM network impacted the performance negatively. This is reflected both in the training loss and the following model performance which is evaluated using the Bilingual Evaluation Understudy (BLEU), the BLEU-NLTK, a variant of the BLEU score, and lastly the Translation Edit Rate (TER). Using LoRA in the MLP layer, and not exclusively in the attention layer, the number of trainable parameters is increased since the entirety of the MLP layer is no longer unchanged for retraining. This means that along with the worse model performance the models take longer to train. To further identify the issue with LoRA in the MLP layer, LoRA was applied to a pure MLP network trained and retrained on the MNIST data set. This did not result in lower accuracy, hinting that the issue arises with LoRA in the MLP layer of transformers and not for MLP layers in generel.

1 Introduction

With the arrival of Large Language Models, followed the challenge of making them broadly accessible. LLMs can be very computationally expensive due to the vast number of parameters and the size of the utilized data set. With these hindrances, the training of LLMs has primarily required access to significant computational resources which has limited the overall accessibility. As a result, various methods of compression has emerged aiming to minimize the computational resources whilst avoiding any significant loss in performance. Additionally, when using pre-trained LLMs like the GPT-networks [9], BERT [4] and Llama [10], and retraining them for specific tasks, methods like full fine tuning of all the parameters require full re-computation of all parameters which again is computationally heavy.

As part of this work, currently available methods of compression including pruning and quantization are reviewed. The main focus is however the Low Rank Adaptation (LoRA) [7], which we have chosen to expand upon. In the work by Hu et al. [7] LoRA is implemented in the attention layer of the transformer. We build upon this by investigating the effect of implementing LoRA in the MLP layer of the transformer, whilst preserving LoRA in the attention layer as well. For comparison we train LLM's both with and without LoRA in the MLP layer and we also retrain a pure MLP network using LoRA.

Preprint. Under review.

2 Related Work

2.1 Pruning

Pruning is a method used to minimize the number of trainable parameters by removing redundant weights using a set threshold [6]. It serves as a valid approach to reduce the complexity of a model, improve efficiency, and even act as a normalizer to prevent overfitting.

While in computer vision, you can prune about 95% of parameters without severe performance degradation, the pruning capability for transformers trained on NLP data drops to around 30%. This is primarily because transformers include an attention layer, which cannot be easily pruned in the same manner as the CNN layers traditionally are. The percentage further drops to about 5% after the emergence of outlier features when using LLM.int8() by Dettmers [1], because the weaker features are already given less weight, which can undermine the effectiveness of pruning [1].

2.2 LLM.int8()

The LLM.int8() method proposed by Dettmers et al. [2] addresses the challenge of the accuracy drop observed with transformers with a large number of parameters when using traditional 8-bit quantization methods due to the emergence of outlier features.

To mitigate this issue, the LLM.int8() method separates the outlier features and weights, which are then multiplied in 16-bit, while all other values are multiplied in 8-bit. This is achieved through 8-bit vector-wise multiplication, scaled by row and column-wise absolute maximum values of the input features and weights. Subsequently, the multiplication outputs are dequantized by the outer product of normalization constants, and both outlier and regular outputs are accumulated in 16-bit floating-point outputs.

This method presents a promising approach to effectively reduce memory usage in LLMs while maintaining accuracy, thus enhancing their efficiency and applicability in various tasks and environments.

2.3 Low-Rank Adaption

Low-Rank Adaption (LoRA) by Hu et al. [7] is a method to reduce the number of trainable parameters in Large Language Models. This approach offers an alternative to full fine-tuning on pre-trained models while achieving comparable performance to other compression methods and even full fine-tuning. LoRA reduces memory requirements by utilizing a small set of trainable parameters without updating the full model parameters, which remain fixed during retraining.

The pre-trained parameters are frozen during the retraining process, and new parameters are introduced. The number of new parameters is minimized by controlling the *rank*, which acts as a bottleneck. This significantly decreases the trainable parameters from d^2 to $2dr$, where $r \ll d$. After retraining, the new parameters are multiplied and added to the frozen parameters.

In contrast to other compression methods, LoRA training converges to full fine-tuning as the rank increases, while adaptor-based methods converge to an MLP and prefix-based methods to a model that cannot handle long input sequences.

LoRA introduces a novel method for retraining pre-trained models for new tasks, such as overcoming language and cultural barriers. This method offers promise for enhancing the adaptability and efficiency of LLMs in diverse application scenarios.

3 Method

Inspired by the work by Hu et al. [7] we investigate implementing the LoRA method in the MLP layer of the large language model Generative Pre-Trained Transformer 2 network by OpenAI [9].

Utilizing the published code [1] by Hu et al. [7, 8] we implement the LoRA method in the MLP layer while keeping the LoRA implementation in the attention layer. This involved modifying the forward pass of the MLP layer to add a LoRA layer as well as replacing outdated packages and functions.

[1] `https://github.com/microsoft/LoRA`

Table 1: Default hyperparameters from [7, 8], which we used in this work

Training	
Optimizer	Adam W
Weight Decay	0.01
Dropout Prob	0.1
Batch size	8
# Epoch	5
Warmup Steps	500
Learning Rate Schedule	Linear
Label Smooth	0.1
Learning Rate	0.0002
Adaptation	$r_q = r_v = 4$
LoRA α	32
Inference	
Beam Size	10
Length Penalty	0.8
no repeat ngram size	4

3.1 Training circumstances and computing resources

We used M1 and M2 chips for local implementation but transitioned to Google Colab paid service due to the slow training performance experienced. Utilizing A100 GPU, we accelerated the training and testing process by over 90%. The data set used in this work is from the WebNLG Challenge 2017 [5], comprised of text triplets that map to English sentences. The detailed overview of the hyperparameters etc. employed in our work is delineated in Table 1.

The GPT-2 network is a transformer-based architecture composed of a stack of transformer blocks containing an attention layer and a MLP layer. The attention layer enables the model to focus on different words in a sentence and capture the relationships between them, while the MLP layer learns the non-linearities in the input. We implemented the LoRA method on both layers, with and without LoRA in the MLP layer, to investigate its impact on the model's performance and explore the interaction between the LoRA layer and the existing components of the network. We implemented small and medium GPT-2 networks to compare the results of different sizes of networks.

To evaluate the performance of our model, we introduced several standard metrics commonly used in natural language processing evaluations. These include perplexity, which measures the model's ability to predict the next token in a sequence, BLEU (Bilingual Evaluation Understudy), which assesses the similarity between the generated text and reference sentences, BLEU-NLK (Natural Language Knowledge), which considers semantic similarity in addition to syntactic similarity, and TER (Translation Edit Rate), which quantifies the similarity between the generated text and human-written translations through edit operations such as insertions, deletions, and substitutions. These metrics collectively provide a comprehensive assessment of the quality and fluency of the generated text outputs.

3.2 Pure MLP Experiment

In an attempt to further investigate the impact of LoRA implementation, we designed and conducted a supplementary experiment using pure MLP network. In this experiment, we focused exclusively on the MLP layer without the attention layer. We trained the pure MLP model using the MNIST image dataset in the Google Colab environment. After training, we evaluated the performance of the pure MLP model with LoRA implementation using accuracy. The findings from this experiment contribute valuable insights into the efficacy of LoRA implementation in various neural network architectures, showing its potential benefits and limitations.

Table 2: Compression Ratios for some of the trained models

Small network without LoRA in MLP			
Rank	4	8	16
Trainable Parameters	147,456	294,912	589,824
Compression Ratio	0.12%	0.24%	0.47%
Small network with LoRA in MLP			
Rank	4	8	16
Trainable Parameters	331,776	479,232	774,144
Compression Ratio	0.27%	0.38%	0.62%
Medium network with and without LoRA in MLP			
LoRA in MLP	No	Yes	
Trainable Parameters	393,216	884,736	
Compression Ratio	0.11%	0.25%	

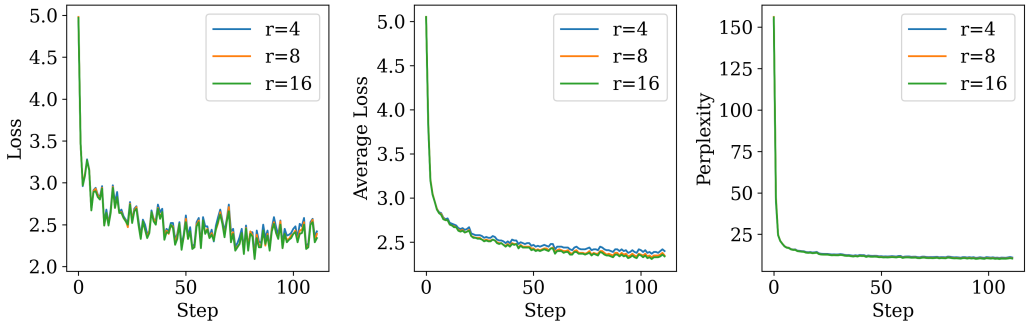

Figure 1: Small GPT-2 network retrained without LoRA in the MLP layer

4 Results

4.1 Compression Ratio

For different network configurations, the compression ratio was computed. As shown in Table 2, all the compression ratios are below 1%, indicating a significant decrease in the number of trainable parameters after introducing the LoRA layer. Showing a successful parameter reduction after the implementation of LoRA.

4.2 Training Loss and Perplexity

Figure 1 provides the training loss and perplexity obtained from the training process with the LoRA implementation only in the attention layer. The Figure showcase decreasing loss and perplexity in all ranks.

Figure 2 provides the training loss and perplexity obtained from the training process with the LoRA implementation in both the attention and MLP layer. From Figure 2 we witness generally nicely decreasing loss and perplexity curves for rank 8 and 16. However, for rank 4 we witness a sudden spike and return to the initial loss levels.

For comparison, we plot the loss and perplexity for the small and medium GPT-2 network with rank 4. As shown in Figure 3, it is clear that the lowest loss is achieved by the networks without LoRA in the MLP layer.

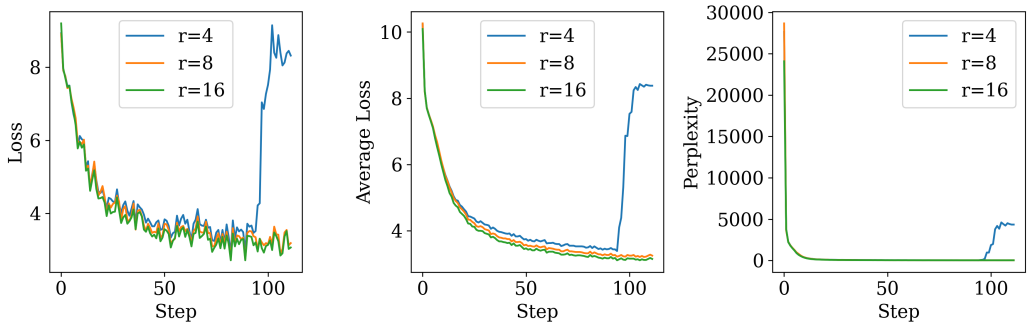

Figure 2: Small GPT-2 network retrained with LoRA in the MLP layer

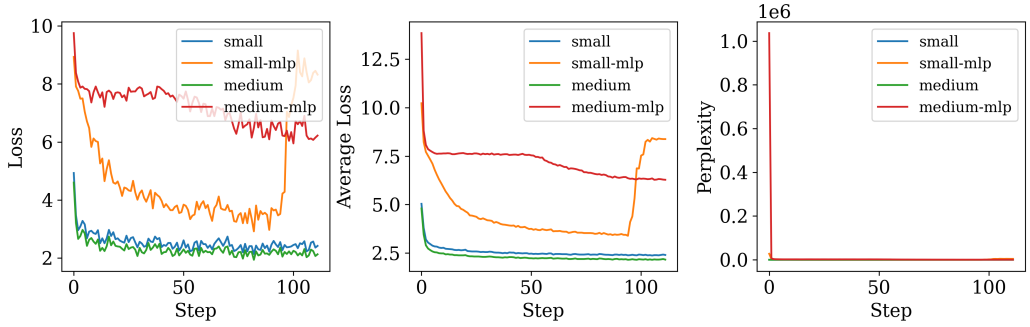

Figure 3: Small and Medium GPT-2 network retrained with and without LoRA in the MLP layer

4.3 Performance Evaluation

The BLEU, BLEU-NLTK and TER scores of models with and without LoRA in the MLP layer are plotted as a function of increasing rank in Figure 4. From this we witness that the BLEU and BLEU-NLTK scores do not keep increasing as the rank increase. We also see that the model with LoRA in the MLP layer yield lower scores than its counterpart with LoRA just in the attention layer. In regard to the TER score, lower is better, so here we also witness that the model with LoRA just in the MLP layer outperforms the model which also has LoRA in the MLP layer.

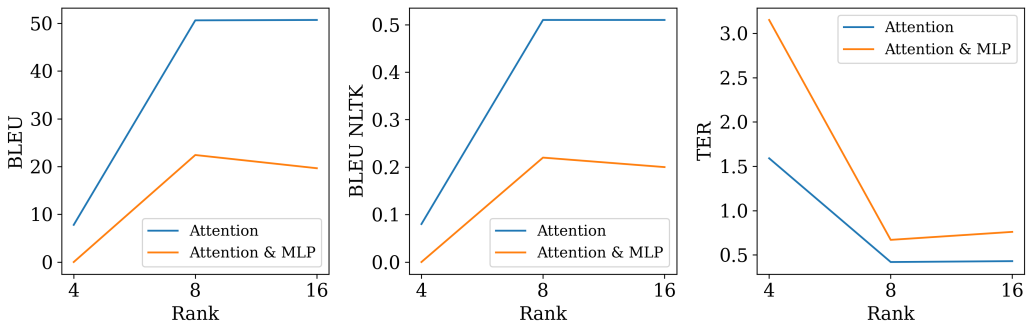

Figure 4: BLEU, BLEU-NLTK and TER scores of Small GPT-2 Networks

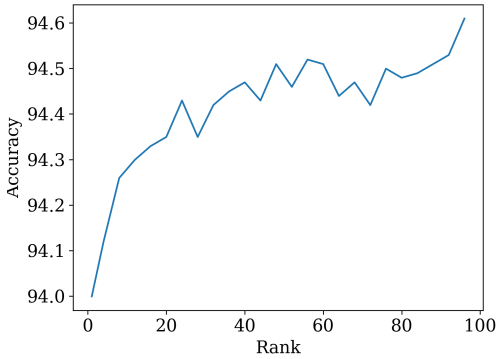

Figure 5: LoRA Implementation in a Pure MLP Network

4.4 LoRA in a Pure MLP Network

The resulting accuracies obtained from the supplementary experiment of the LoRA implementation in a pure MLP network are plotted against increasing rank in Figure 5. The experiment evaluated the classification accuracy of the pure MLP network through different rank sizes. As shown in the Figure, the accuracy generally increases as the rank size increase.

5 Discussion

In reviewing the literature, the implementation of LoRA in the MLP layer remained an area yet to be thoroughly investigated. Thus, this study aimed to assess the impact of introducing LoRA to the MLP layer. Integrating LoRA into the MLP layer adds additional parameters to the model, thereby increasing its complexity. This augmentation raises concerns regarding the potential for overfitting, as the interference of LoRA with the MLP layers' non-linear transformations may impede the model's ability to capture intricate patterns in the data. However, as we progressively increased the rank from 4 to 8 and 16, the issue of overfitting appeared to be mitigated, as the higher rank sizes strike a better balance between model complexity and its capacity to learn from input data. This reduction in overfitting risk allows the model to better utilize the additional parameters introduced by LoRA. Nonetheless, our results indicate that networks without LoRA in the MLP layer consistently achieve lower loss and higher performance, suggesting that the implementation of LoRA in the MLP layer may not yield the desired effectiveness that was initially anticipated. Additionally, the lack of performance improvement as the rank size increases suggests the need for further investigation into the underlying issues impacting the model.

To elucidate the reasons behind the negative impact of LoRA when applied in the MLP layer, we designed a supplementary experiment involving the implementation of LoRA on a pure MLP network using the MNIST image dataset in the Google Colab environment. Interestingly, we observed no performance degradation in this experiment, which corroborates our previous finding that LoRA in the MLP layer interferes with the transformer's ability to learn the intricate structure of the corpus. Furthermore, we observed a slight increase in performance as we increased the rank size. However, the increment was less than 0.5% from rank 1 to rank 100, which is lower than the 1% increment observed when applying LoRA with rank 1 on the pre-trained model. Consequently, it appears that the model does not benefit significantly from a larger rank size. Therefore, we recommend applying a considerably smaller rank size to reduce computational expense without sacrificing performance.

Throughout our experiment, we needed to alter the code and replace some outdated packages, as well as functions that are no longer compatible with MacOS and Google Colab environments. These alterations were essential to ensure the execution of our experiment, but they also might potentially cause discrepancies between our results and the literature under the same configuration.

6 Conclusion

This work set out to review existing methods of compressing LLMs such as pruning, LLM.int8(), and LoRA in order to make them more computationally accessible. We then proposed a new method investigating the effect of implementing LoRA in the MLP layer of a transformer network. Our result show that this implementation degrades the performance compared to just applying LoRA in the attention layer.

Despite its negative impact on performance when applied in the MLP layer of the transformer, LoRA remains a valuable model compression method due to its ability to reduce trainable parameters. It is worth noting that while LoRA can be implemented in any network layer involving matrix multiplication, its usage in the MLP layer of a transformer is not recommended according to this work.

6.1 Future Work

Further research might explore the possibility of implementing the LoRA method in the LayerNorm layers or the biases, as this could provide additional flexibility and control over the retraining process. Nevertheless, the potential for performance degradation, as observed when applying it in the MLP layer, is a concern. Thus, a extensive hyperparameter search is needed to determine the baseline of the study before future exploration.

Another possible compression method to investigate is QLoRA by Dettmers et al. [3] which combines the benefits of LLM.int8() [2] and LoRA [7].

References

[1] T. Dettmers. Llm.int8() and emergent features. `https://timdettmers.com/2022/08/17/llm-int8-and-emergent-features/`, Aug. 2022.

[2] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

[3] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

[5] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini. Creating training corpora for nlg micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1017. URL `http://www.aclweb.org/anthology/P17-1017`.

[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] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021. URL `https://arxiv.org/abs/2106.09685`.

[8] 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 *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

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

[10] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.