To the Short Context and Beyond: Modelling Long Sequences with LongLoRA

Elisa Forcada Rodríguez
University of the Basque Country
eforcada001@ikasle.ehu.eus

Abstract

This paper investigates three different methods for fine-tuning LLMs pre-trained models. The LongLoRA technique, an extension of the LoRA method, shows superior efficiency and reduced computational and time costs over the other two methods, full fine-tuning and LoRA. In order to demonstrate this, three experiments are designed and proposed, to compare the technique's performance on the perplexity metric.

1 Introduction

Large Language Models (LLMs) have become increasingly important in recent years, due to their ability to understand natural language and solve complex tasks (Zhao et al., 2023): the ChatGPT family is a clear example of the fame and popularity that these models have achieved in this short period of time. Regarding their structure, LLMs typically have the architecture of Transformer models, which contain millions or even billions of parameters, and have been trained with massive corpora of text data.

During an initial phase, LLMs are usually trained with these large corpora to perform general tasks (such as sequence classification or prediction of a word within a sentence). Therefore, if the same model is to be used for a new, more specific task while maintaining its original good performance, a new more concrete training must be conducted with training data adapted for the new assignment. This process can be accomplished in several ways, each with its own advantages and limitations.

This paper aims to explain and illustrate the differences between three different possible methods to do this process: full fine-tuning, LoRA, and LongLoRA. For this purpose, these techniques are described, with their own benefits and limitations. Then, three experiments are designed and explained, in order to obtain comparisons between the performance of these methods.

2 Full fine-tuning

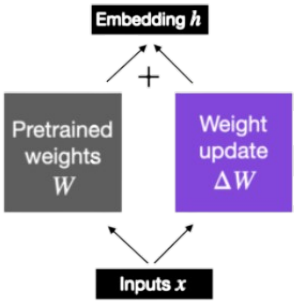

Figure 1: Example of the process of weight adaptation of a matrix during full fine-tuning

Fine-tuning is a process that adapts a pre-trained language model to multiple downstream applications (Hu et al., 2021). This method is simpler and faster than creating and training a model from scratch, considering the large number of parameters that these architectures contain. However, it has some limitations: during full fine-tuning, all the parameters of the pre-trained model are updated. As a result, the new model contains the same parameters as the original one, which can result in high computational and time costs, as well as other potential issues, like I/O bottlenecks.

To address these limitations, more efficient adaptation methods have been developed by experts, such as adding adapter layers (Houlsby et al., 2019) or optimizing certain forms of input layer activations (Li and Liang, 2021). Although these solutions have been proven to be more efficient than full fine-tuning, they still have limitations that can be problematic in large-scale, latency-sensitive scenarios.

Adapter layers introduce inference latency, even though they usually have a relatively small number of parameters: even with less than 1% of the number of parameters of the original model, introducing these adapters can increase the required

inference time, due to the need for these layers to be processed sequentially. Thus, the available hardware parallelism, which is one of the most used hardware method to keep the latency low, is not employed (Houlsby et al., 2019).

The other option, layer activations optimisation, exemplified by prefix tuning, has also its own limitations and difficulties: optimising a prefix directly is challenging, and finding the perfect configuration for the prefix to consistently improve model performance is very difficult. Additionally, reserving part of the sequence length for this adaptation restricts the amount of information the model can process for the main task (Li and Liang, 2021).

3 LoRA

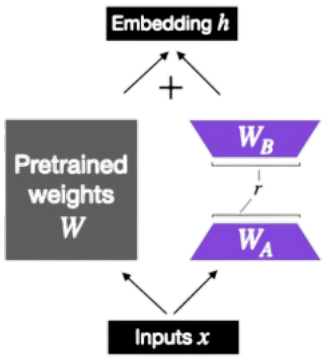

Figure 2: Example of the process of weight adaptation of a matrix applying LoRA

LoRA (an acronym for Low Adaptation Rank) is a proposal that aims to overcome the limitations of the previously explained methods. It is based on Aghajanyan et al. (2020) and Li et al. (2018), which demonstrate that the over-parametrized models learned actually reside on a low intrinsic dimension. As mentioned earlier, most LLMs have a Transformer architecture, consisting of dense layers with matrices of very large dimensions and full rank (i.e. all their columns are linearly independent). Despite this, it has been claimed that during model adaptation to a specific task, the dimensionality of these matrices can be reduced without affecting the model's performance.

LoRA enables the indirect training of the dense layers of a neural network in a computationally and time-efficient manner by decomposing the dense layer matrices into two smaller matrices with much lower rank, while keeping the pre-trained weights frozen (Hu et al., 2021).

This adaptation is demonstrated in the following formula: to modify a pre-trained matrix $W_0 \in \mathbb{R}^{d\times k}$, the matrix $\Delta W x$ (which has the same dimensions as $W_0 x$) is reduced, by being decomposed into two smaller matrices, $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$, with rank $r \ll \min(d, k)$.

$$\mathbf{h} = W_0 x + \Delta W x = W_0 x + BAx$$

At the start of training, A is initialised using a random Gaussian distribution, while B is set to 0. $W_0 x$ stays frozen, unchanged. This means that the resulting weight update matrix $\Delta W x$, which is responsible for adapting the model to a new task, is also initialised to 0. LoRA reduces the computational and time cost of full fine-tuning by decreasing the number of parameters that need to be updated during training: as stated before, full fine-tuning updates all parameters of the original model, which can be computationally and time expensive. Furthermore, LoRA does not require sequential processing like adapters, which can increase inference latency: this is because LoRA does not need the accumulated grandient updates to the weight matrices to have a full range during adaptation (Hu et al., 2021).

Additionally, LoRA allows for quick switching to another task, as the original W_0 matrix can be easily recovered by subtracting BA from the resulting matrix: LoRA offers a fast operation with minimal memory overhead, ensuring no additional latency during inference. This is not the case with full fine-tuning or other adaptation methods (Hu et al., 2021).

In summary, the most significant advantage of LoRA over full fine-tuning is the significant reduction in memory and storage usage. This is evident in the case of GPT-3 175B, where VRAM consumption during training decreases from 1.2TB with full fine-tuning to 350GB with LoRA and **r=4**. It is important to note that even with a small range, such as 4 or 6, the performance will be similar to that of full fine-tuning. Also, LoRA allows for switching between deployed tasks at a low cost and without introducing additional latency, which sets it apart from other proposals (Hu et al., 2021).

On the other hand, LoRA has limitations in grouping inputs from different tasks, particularly when they require different settings (i.e. different B and A). Attempting to combine these inputs in a single process can complicate input management and

increase model complexity. Also, absorbing the settings (B and A) into the W weights to avoid delays in inference may further exacerbate this complexity (Hu et al., 2021).

4 LongLoRA

As previously mentioned, LoRA enables efficient fine-tuning and reduces computational and time costs. However, it may encounter an issue with context lengths. LLMs are typically pre-trained with a predetermined context size, such as 2048 tokens for Llama (Touvron et al., 2023a) and 4096 tokens for Llama2 (Touvron et al., 2023b).

This predetermined length can restrict the performance of models in various tasks, and training a model from scratch with a lengthened context is prohibitively expensive (Chen et al., 2023b). That is why several approaches have been developed to extend the context of a pre-trained LLM: one example is the method called Position Interpolation, which modifies rotary position encoding (Chen et al., 2023a). Another example is FOT, which uses contrastive learning for training (Tworkowski et al., 2023). However, these techniques have the limitation of high computational costs.

Regarding LoRA, if it is used for fine-tuning and context lengthening, the model's performance decreases significantly. The gap between LoRA and full fine-tuning is substantial as the target context length increases. Even with an expanded range, LoRA's performance is far from reaching the same level as full fine-tuning (Chen et al., 2023b). The techniques mentioned earlier, including full fine-tuning, have shown significantly higher performance in this scenario. However, they come at the cost of losing the advantages obtained in LoRA.

That is why LongLoRA was created: it is a method based on LoRA (Chen et al., 2023b) that aims to achieve good model performance with an elongated context size while maintaining or even improving the advantages of LoRA. To achieve this, two concepts are proposed: shifted sparse attention (S^2-Attn) and LoRa+. These concepts aim to improve the efficiency and effectiveness of any model.

4.1 S^2-Attn

Shifted sparse attention (S^2-Attn) is a combination of two attention patterns. One is based on short attention, which involves dividing a long context input into several groups in self-attention. For example, if there is an input of 8000 tokens and it is desired to use it with a model with a pre-trained context of 2000 tokens, this input will be split into four equal groups of 2000 tokens each. This approach is efficient, but it does not increase the context of the models as there is no exchange of information between the different groups. Therefore, as the context length grows, the models have more perplexity.

The second approach involves shifting the group partition by half the group size in half of the attention heads. Continuing from the previous example, the input of 8000 tokens would be shifted by 1000. This means that the first attention group would start from token 3000 to 5000, while the first and last 1000 tokens would belong to the same group.

Afterwards, both patterns are combined and used in each half of the self-attention heads respectively. This method does not add computational costs and allows information to flow between the different groups. Additionally, the obtained performance is similar to that of the standard attention baseline. S^2-Attn is easy to implement, requiring only two steps: moving tokens within the attention heads and transposing features from the token dimension to the batch dimension. This can be achieved with just two lines of code.

Additionally, S^2-Attn overcomes the limitations of existing attention methods that are not optimal for fine-tuning tasks requiring extensive context understanding. It enables efficient fine-tuning and supports full-attention testing. The model can be tested using standard full attention without compromising its performance, even if a different form of attention is used during fine-tuning.

However, shifting may introduce potential information leakage, which can be prevented through a small modification on the attention mask.

4.2 LoRA+

LoRA adapts attention weights in Transformers architectures while freezing the other layers, including MLP and normalization layers. However, adapting only the attention weights does not work for long context extension.

Therefore, LoRA+ proposes to apply the LoRA method and fine-tune not only the attention weights, but also the normalization and embedding layers, as they are crucial for low-rank long-context adaptation (Chen et al., 2023b).

4.3 LongLoRA benefits

LongLoRA enables the benefits of full fine-tuning and LoRA in long context sequences scenarios. Its two implementations provide comparable performance to full fine-tuning, with reduced cost and time due to the extended application of LoRA. Among the three methods studied in this paper, LongLoRA is the most efficient and comprehensive, as it combines the advantages of the other two methods.

5 Experiments

Three experiments were conducted to demonstrate the differences between the three methods described above. All the related code is available on Github: https://github.com/elisafr00/FinalProjectDL4NLP.

Running original code locally

For the first experiment, the original LongLoRA project[1], available on its creator's Github page, was run locally by using two coding environments, Visual Studio Code and Spyder. The implementation of this option was not possible due to errors caused by uninstalled packages. After installation, the packages were found to be incompatible with each other[2].

Running transformed original code in the cloud

After it was demonstrated through various methods that the original project could not be executed locally, attempts were made to run it on cloud platforms such as Jupyter Notebook and Google Colab. However, in order to run the code on these platforms, it was necessary to convert the files format from .py to .ipynb, as these environments only accept this file type.

The p2j (https://github.com/remykarem/python2jupyter) library was used to perform this transformation on all files individually[3]. After making these transformations, an attempt was made to run the adapted scripts. This time, there were fewer problems related to package installations, which were easily solved. However, an error occurred when importing a module from the project, called "replace_llama_attn". Despite the module's existence and its apparent correct configuration, an error related to it appeared when running the demo script[4].

Using pre-trained models from Hugging Face

To avoid issues caused by the original code, the experiment line was switched to pre-trained models with full fine-tuning, LoRA and LongLoRA, stored in Hugging Face. This approach was proven to be effective in resolving the problems observed with the original code.

To achieve this, a script in Google Colab was created, which is available on Github: https://github.com/elisafr00/FinalProjectDL4NLP/tree/main/hugging%20face. This script uses the **Transformers** library and the **pipeline** function of Hugging Face to load any model stored in the repository. After understanding the functionality of this library, a function was developed to calculate the **perplexity** of each model (an evaluation metric explained in more detail below). This function was then tested by using example models and datasets to ensure its correct operation.

Finally, the selected models and dataset (explained in the following sections) were loaded. However, the large size of these models caused Google Colab to run out of RAM, even with the **paid** Google Colab Pro option, which increased the free RAM from 12.7 GB to 51.0 GB. Therefore, this script could not be executed due to a computational problem. In any case, its objectives were clear, and its correct functioning was demonstrated with other example models. Therefore, the following sections will focus on this last experiment.

5.1 Models

To compare the three techniques, three similar models hosted on Hugging Face were chosen. Each model was trained with one of the different methods to be compared. These models were:

- Llama2-7B with full fine-tuning[5]
- Alpaca-7B with LoRA[6]

[1]The original code can be found at https://github.com/dvlab-research/LongLoRA

[2]This cloned repository is also available in the Github repository of this project: https://github.com/elisafr00/FinalProjectDL4NLP/tree/main/original%20code

[3]The files that have been converted to .py can be found on Github: https://github.com/elisafr00/FinalProjectDL4NLP/tree/main/.ipynb%20code

[4]The files associated with this error can be found on Github: https://github.com/elisafr00/FinalProjectDL4NLP/tree/main/failed%20.ipynb%20experiments

[5]Llama2-7b with full fine-tuning available at https://huggingface.co/Yukang/Llama-2-7b-longlora-8k-ft

[6]Alpaca-7B with LoRA available at https://huggingface.co/TehVenom/Pygmalion_AlpacaLora-7b

- Llama2-7B with LongLoRA[7]

Both the first and third models, based on Llama2 (Touvron et al., 2023b), were selected because they appear in the LongLoRA paper (Chen et al., 2023b) and were used in the official experiments to obtain statistics on the differences between the three techniques.

The second model, based on Alpaca (Taori et al., 2023), was fine-tuned with LoRA. All models were pre-trained with 7 billion parameters, and the first and the third had the same predefined context length of 8000 tokens. Therefore, it could be concluded that these models were very similar, with the only difference being the fine training. This would have allowed to draw accurate conclusions about the performance differences of the three techniques.

But, as previously stated, the size of these models rendered them unsuitable for use on standard computing platforms like Google Colab or Jupyter Notebook. However, commercial platforms could run these models, and therefore provide valuable metrics and information for this project.

5.2 Dataset

It is important to note that the main objective of the experiment related to Hugging Face was to extract metrics from three different models on a given dataset with different sequence lenghts inputs. The focus was on testing their performance, rather than training them. Therefore, to achieve this objective, the PG-19 dataset was selected, specifically the testing partition, which was also used by the authors of the LongLoRA paper in their experiments (Chen et al., 2023b).

PG-19 is available at Hugging Face, just like the selected models: https://huggingface.co/datasets/pg19. As its developers state, this dataset consists of various books extracted from the Project Gutenberg library, along with metadata on book titles and publication dates (Rae et al., 2019).

Despite its considerable size, this dataset could be loaded without any issue in Google Colab Pro. The only drawback was the time it took to load completely, which was over an hour.

5.3 Perplexity

In order to compare the performance of the three models, it was decided to use perplexity as the chosen metric, in line with the experiments presented in LongLora's paper (Chen et al., 2023b).

Perplexity is a metric that refers to the "surprise" or perplexity of an LLM when confronted with a dataset it has never seen before. In statistical terms, this measure calculates how well the model can predict certain outcomes. Consequently, the lower the perplexity of a model, the better its performance: a low perplexity indicates that the model makes very accurate predictions, and therefore performs well on the given dataset. In the context of NLP, perplexity is usually calculated from the cross-entropy loss. This type of perplexity can be better understood with the following formula:

$$\textbf{Ppl} = 2^{H(y,\hat{y})}$$

Where $H(y, \hat{y})$ is the cross entropy between the actual labels y and the model predictions $\hat{y}$. As previously mentioned, a function was developed to accurately calculate this metric on the available models and dataset in Hugging Face. However, due to a RAM issue when loading the models, the function could not be executed on the target models, but only on the example ones.

5.4 Expected results

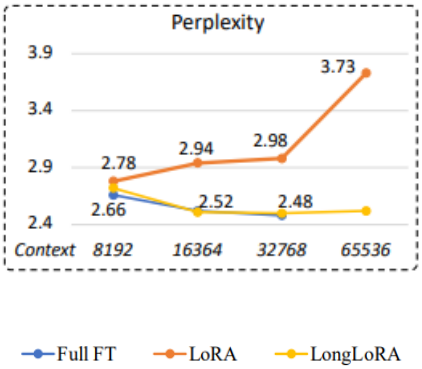

Figure 3: Perplexity results from LongLoRA paper

As previously mentioned, three experiments were attempted, althought none of them were completed. The first two were unsuccessful due to incompatibilities with libraries or modules that were not properly developed by the creators of the original experiment. The third experiment was hindered by a limitation in RAM memory when loading models in online environments, such as Google Colab.

However, the experiments' objectives aligned with those proposed in the original LongLoRA paper (Chen et al., 2023b): the perplexity of the three

[7]Llama2-7B with LongLoRA available at https://huggingface.co/Yukang/Llama-2-7b-longlora-8k

types of training was going to be obtained and compared, along with information on the computational and time cost of each model. In order to do this, the input sequences would have had a context length of varying sizes.

The main hypothesis of this project is that, in terms of perplexity, models with full fine-tuning and LongLoRA would have performed similarly. However, LoRA would have been significantly less efficient, and its performance would have deteriorated as the context length increased. This is demonstrated in Figure 3 of the paper (Chen et al., 2023b), which displays the experimental results of LongLoRA, indicating that LongLoRA and full-fine-tuning have comparable performance.

6 Conclusions

This paper explores three techniques for fine-tuning pre-trained LLMs: full fine-tuning, LoRA, and LongLoRA. In order to compare several models trained with the three techniques in different scenarios with varying context lengths by using the perplexity metric, three experiments are designed and proposed.

Although there were certain limitations preventing the fulfillment of these three experiments, it is possible to draw conclusions based on the theoretical concepts and results presented in the explained literature: LongLoRA is the most efficient and profitable method of all three, as it combines the good performance of full fine-tuning when lengthening the predefined input sequence context lenght, with the low computational and temporal costs of LoRA.

7 Discussion

This paper explores three techniques for fine-tuning pre-trained models. It would be interesting to run all the experiments to obtain metrics for precise comparisons. On the other hand, extending the comparison with new models and datasets could reveal new areas for research. Additionally, further research could be conducted on other fine-tuning methods, such as QA-LoRA.

References

Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023a. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023b. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. 2019. Compressive transformers for long-range sequence modelling.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

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. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, and Xavier Martinet. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. 2023. Focused transformer: Contrastive training for context scaling.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.