Final Report For Group 5

Yaqi Hu yaqihu@usc.edu
Gonglin Chen gonglinc@usc.edu
Libang Xia libangxi@usc.edu
Zhaojin Yin zhaojiny@usc.edu
Luoyuan Zhang luoyuanz@usc.edu

1 Introduction

1.1 Paper Introduction

The paper, Linearizing Transformer with Key-Value Memory, an innovative transformer variant designed to address the computational inefficiencies of transformer. As the vanilla transformer suffers from quadratic computational overhead (See Fig.1), MemSizer proposes a solution that combines low-rank projections and memory-efficient recurrent-style generation, to solve it.

It introduces a key-value memory layer to replace the multi-head attention layer, simplifying the design by focusing more on the memory values while using fixed-sized, input-independent parametric matrices as memory keys (See Fig.2). This approach retains linear computation time and constant memory complexity, and significantly reduces multi-head computation complexity.

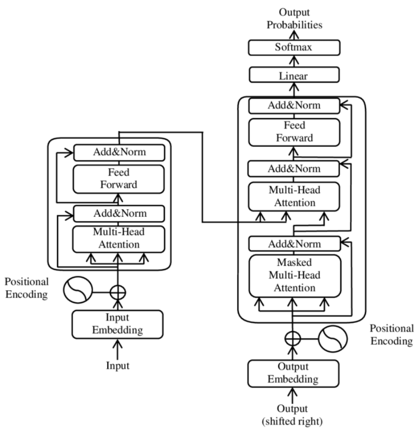

Figure 1: Transformer Architecture
Source: Wikipedia

1.2 Project Introduction

Our project mainly focus on the reproducing and extension of the MemSizer model, with replacing core of a question and answer (Q&A) with Memsizer previously built using transformer.

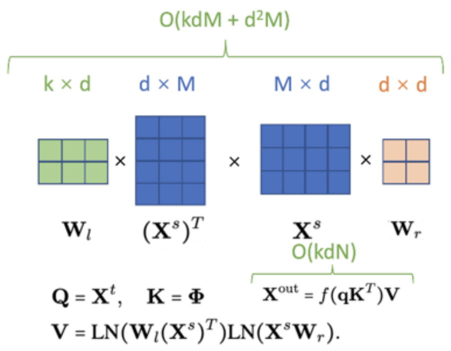
$O(kdM + d^2M)$
$k \times d$
$d \times M$
$M \times d$
$d \times d$
$\mathbf{W}_l$
$(\mathbf{X}^s)^T$
$\mathbf{X}^s$
$\mathbf{W}_r$
$O(kdN)$
$\mathbf{Q} = \mathbf{X}^t, \quad \mathbf{K} = \mathbf{\Phi}$
$\mathbf{X}^{\text{out}} = f(\mathbf{q}\mathbf{K}^T)\mathbf{V}$
$\mathbf{V} = \text{LN}(\mathbf{W}_l(\mathbf{X}^s)^T)\text{LN}(\mathbf{X}^s\mathbf{W}_r).$

Figure 2: MemSizer Memory Network

Our primary object is to reproduce and validate the results from the MemSizer paper, a cutting-edge memory optimization tool. Moreover, We extended the paper model to a newly published dataset, OpenOrca, which presents challenges and characteristics different from those in the original used datasets.

The integration process involved adapting MemSizer with our existing Q&A system, with aiming of enhancing the system's performance, particularly in term of GPU memory efficiency and training time/model size, also with response accuracy. Moreover, compare both results, to see the performance of the MemSizer.

1.2.1 Reproducing

For reproducing the paper result, in original paper, the research team contain six datasets from two different NLP research directions. These are too many for us to reproduce, therefore, we select each only one dataset in two tasks. They are:

1. Machine Translation:
   WMT 16 En-De
   https://huggingface.co/datasets/wmt16/viewer/de-en/train

2. Text Summarization:
   XSUM
   https://huggingface.co/datasets/xsum

We get these two datasets from the link provided above. For both original and our new used datasets, we use the standard preprocessing method (so do as the paper author). It contains several steps: Tokenization, Lower-casing, Removing Special Charactersand etc. Just like the homework 1 we did before.

Note: we only use nearly 5 percent of the dataset train set. (Twenty thousand samples) It is limited in the preprocessing step, rather than training. So it doesn't matter with our reproduce. As we have a limited GPU memory usage and we only want to see the training time/GPU usage/speed

After we produced the results for MemSizer, we also run a transformer in same two datasets, and compare their result (GPU memory efficiency, training time/model size, response accuracy) to see how better the MemSizer can achieve. The result is shown in Results&Discussion part.

To highlight the task, we rewrite structure for Memsizer, to let it run. The official repository cannot run directly and we have connected the author, but no one answered us.

1.2.2 Extended Work

We extended the MemSizer to a newly published dataset, OpenOrca (https://huggingface.co/datasets/Open-Orca/OpenOrca). It is a rich collection of augmented FLAN data aligns.

Along with the dataset, it published a paper and their latest model, the first 7B to score better overall than all previous models below 30B. 98% of Llama2-70b-chat's performance, in a completely open 7B. This dataset supports a range of tasks including language modeling, text generation, and text augmentation.

The extended work result with transformer and MemSizer can be seen in Result&Discussion part.

1.2.3 Adapting To Our Q&A system

In order to further observe the performance and performance of MemSizer, we designed an application scenario, the Q&A system. We have write a short Q&A system with transformer, and trained it a math dataset with about 2 million records from math dataset. Then we record the performance of the transformer, and then replace transformer with MemSizer, then we could compare these two models in a real world scenario. (Of course, not very real, as it is a simple task)

The code can been seen in the Github link (It is a public repository): https://github.com/YaqiHu23/CSCI544

Note: the repository contains all the code we used, and the slides.

2 Methods

2.1 Paper Methods

The model described in original paper, with its distinctive key-value memory mechanism. This architecture modifies the standard transformer by introducing a linearized attention mechanism and a key-value memory store. The linearized attention reduces the computational complexity, making it more efficient, especially for longer sequences. From Original attention:

$$Q = X^t X_q + b_q$$

$$K = X^s W_k + b_k$$

$$V = X^s W_v + b_v$$

$$\alpha = softmax(\frac{QK^T}{\sqrt{h}})$$

$$x^{out} = \alpha V$$

To a new attention:

$$Q = X^t, K = \Phi$$

$$V = LN(W_l(X^s)^T)LN(X^s W_r)$$

$$X^{ouyt} = \frac{1}{r}\sum_{i=1}^{r} X^{out}_{(i)}$$

The key-value memory component is a crucial feature of this model. It stores past information that the model can access, improving its ability to handle long-range dependencies. We adhered to the original implementation details, ensuring that the memory component could dynamically store and retrieve information across different layers of the transformer.

Moreover, it used recurrent computation feature. For each generation step i:

$$V_i = \sum_{j=1}^{i} LN(W_l(X_j^s)^T)LN(x_j^s W_r)$$

Model	batch size	Speed (tokens/s)	Memory (GB)	Model size (M)
MemSizer	64	160	10.3	31.61
Transformer	64	132.7	18.2	68.9

Table 1: WMT 16 En-De Comparison Result

Model	batch size	Speed (tokens/s)	Memory (GB)
MemSizer	64	121.6	15.3G
Transformer	64	103.2	34.5G

Table 2: XSUM Comparison Result

Since x_j^s is just the j^{th} row of X^s, V_i can be seen as a rolling-sum matrix:

$$V_i = V_{i-1} + LN(W_l(x_i^s)^T)LN(x_j^s W_r)$$

This steps used cached matrix for computation, avoiding quadratic computation with sequence length.

2.2 Q&A system Methods

We trained both vanilla transformers and MemSizer with hyper parameters 4 for encoder layers and decoder layers, 512 for model dimension and 2048 feed forward. We used AdamW optimizer with learning rate of $1 * 10^{-4}$ with batch size equal to 64. Both model converges with only one epoch on Google Colab T4 GPU.

We use different vocabulary for Question and Answer. The answer is generated from decoder using greedy decode algorithm. We then deploy these two models with Flask for a web interface which can answer users' question about the value of number place. Users can choose model between Memsizer and Transformer.

3 Experiments

3.1 Dataset Description

3.1.1 WMT 16 En-De

It contains a collection of English and German parallel texts. The texts are derived from various sources, mainly news websites and other publications.

It typically includes millions of sentence pairs. The dataset is part of the WMT (Workshop on Machine Translation) 2016 challenge, and it's known for its quality and diversity in terms of sentence structures and vocabulary.

3.1.2 XSUM

It consists of BBC articles and their single-sentence summaries, providing high-quality reference summaries.

It includes tens of thousands of articles and their corresponding summaries, offering a rich resource for training summarization models.

3.1.3 Open-Orca

It supports a range of tasks including language modeling, text generation, and text augmentation.
A data instance in this dataset represents entries from the FLAN collection which have been augmented by submitting the listed question to either GPT-4 or GPT-3.5. The response is then entered into the response field. This is the reason why we choose this dataset.

3.1.4 Our Q&A dataset

The https://github.com/google-deepmind/mathematics_dataset/tree/master math dataset contains a range of question types at roughly school-level difficulty and answers.

We choose to use numbers place in terms of simplity. An example record of the dataset is
Q: What is the tens digit of 89290? A: 9

3.2 Experiment Setup

We reproduce all the result based on the Google Colab, as our computers have limited computational resources, and cannot get enough GPU memory to run the whole model (even some of us don't have CUDA). We get the result with Google Colab Pro, typically V100 GPU. However, with Google Colab, we still facing the CUDA out of memory errors.

3.3 Training Procedure

We Get the MemSizer code from their offical Github repository, https://github.com/jcyk/memsizer. We are facing the challenge that by

Model	Speed (tokens/s)	Memory (GB)	Model Size (M)
MemSizer	110.72	38.1G	50.87M
Transformer	-	-	-

Table 3: OpenOrca Comparison Result

Model	Speed (tokens/s)	Training Time (Minutes)	Model Sizes (M)
MemSizer	9	11 mins	30Mb
Transformer	47	58 mins	122Mb

Table 4: Q&A Comparison Result

official step, we cannot run the program/model. It shows lacking some folder to run the code, but these folders are not dataset files, so we are struggle for it for a long time.

After that, we decided to rewrite all the structure of program, using core code as same before. Finally we could run the program, and get result shown in the Section Result&Discussion.

4 Results&Discussion

4.1 Reproduce

The Reproduce result is shown in Table.1, Table.2. From the table.1, when doing the Machine Translation task, we can see that MemSizer used less GPU memory, with a smaller model size, which shows it can reduce the disk usage and achieve a better GPU memory efficiency. With such great improvement, we can see the speed of the MemSizer has leaded a significant improvement from transformer.

Also, from the table 2, when doing the text summarization task, it can achieve a great efficiency in GPU memory usage. It helps a lot when we have limited GPU memory.

4.2 Extended Work

For the extended work, which shows the similar result like the reproduced result above. However, as the Colab A100 GPU memory limit is 40G, we cannot run it with transformer on Colab, but we do believe that it could train longer, and has a larger model size. (As it is using more CUDA memory). We can better see that even with newly published datasets, and this dataset is mainly used in small parameter LLM scenario, like the one introduced in the dataset webpage, a 7B model which achieved nearly same result with Llama2-70B one. Memsizer may become more helpful to these scenarios, and can help to develop faster, smaller, better model.

4.3 Q&A System

From the training result shown in Table 4. The vanilla transformer takes about one hour to converge. Surprisingly, when we replace the vanilla transformer with Memsizer, it only take about ten minutes to converge. From the results, it shows again that the MemSizer is efficient in memory usage and can save training time. It may help to build a complex Q&A system like AI agent faster, smaller in the future.

5 Conclusions

To Draw a conclusion for our project, we first reproduced the MemSizer result with two datasets introduced in the paper. (Only using small piece of them). However, the original code cannot be run, so we rewrite their structure, deleted things that useless, and successfully run it. Then we also reproduced transformer result with these two datasets, and compare them in the result part.

Moreover, we extend the MemSizer to a newly published dataset, and compared result with transformer. To better show the improvement, we write a tiny Q&A system as real world application, it also shows the better performance of MemSizer.

From the result we get in two part of our project, we can see that MemSizer do have a great GPU memory usage efficiency and can achieve shorter training time, with smaller model size. It show potential helpings to LLM, as they suffer from long training time, and large model size, also shows in the OpenOrca result. It highlighting the potential of MemSizer in optimizing transformer-based system. The result also offers valuable insights into the adaptability of MemSizer across different tasks with old structure, and it's implications for future application in similar AI-driven systems.

6 Reference

1. Attention is all you need
https://browse.arxiv.org/pdf/1706.03762.pdf
2. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension
https://aclanthology.org/2020.acl-main.703.pdf
3. Linearizing Transformer with Key-Value Memory
https://arxiv.org/abs/2203.12644
4. Generating Long Sequences with Sparse Transformers
https://arxiv.org/abs/1904.10509
5. OpenOrca
https://huggingface.co/datasets/Open-Orca/OpenOrca
6. WMT 16 En-De
https://huggingface.co/datasets/wmt16/viewer/de-en/train
7. XSUM
https://huggingface.co/datasets/xsum
8. How to train a transformer, https://huggingface.co/learn/nlp-course/chapter1/4
9. Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, et al.. 2016. Findings of the 2016 Conference on Machine Translation. In Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers, pages 131–198, Berlin, Germany. Association for Computational Linguistics.
10. Saxton, Grefenstette, Hill, Kohli. "Analysing Mathematical Reasoning Abilities of Neural Models." arXiv, 2019, arXiv:1904.01557.

7 individual Contributions

Final Report Writing: Mainly by Yaqi Hu, Libang Xia, Gonglin Chen revised and reviewed.

Project Processing:

1. Reproduce: Libang Xia, Zhaojin Yin, Yaqi Hu
2. Extend Paper: Yaqi Hu
3. Replace Module Work:
   Gonglin Chen

Final Presentation:
Presentation Person: Zhaojin Yin, Luoyuan Zhang
Slides Writing: Yaqi Hu, Libang Xia, Gonglin Chen, Zhaojin Yin, Luoyuan Zhang