Natural Language Processing (CS5803) Knowledge Editing and Machine Unlearning for Large Language Models

Course Instructor : Dr. Maunendra Sankar Desarkar

Preethi G - AI23MTECH14005
Department of Artificial Intelligence
Indian Institute of Technology, Hyderabad
ai23mtech14005@iith.ac.in

***Abstract*—The remarkable capabilities of Large Language Models (LLMs) in understanding and generating human-like text are hindered by their intensive computational requirements for training, compounded by the need for frequent updates to stay relevant in a dynamic world. Addressing this challenge, I performed two state-of-the-art approaches for Knowledge Editing (lightweight modifications) in LLMs: ROME and MEMIT to ensure continued performance and relevance across different domains.**

I. Introduction

Large Language Models (LLMs) have proven to be very efficient at processing and producing text that resembles that of humans. Two major drawbacks of these LLMs are the large amount of computational power needed for training and the need for regular updates to stay current in a world that is changing quickly limit their usefulness. Knowledge Editing methods for dynamically modifying LLMs are becoming more and more popular as a solution to this problem, as they guarantee consistent performance and flexibility in a range of contexts. In this project, I have investigated the state-of-the-art methods for real-time LLM Knowledge Editing that preserve performance quality while adjusting to specific contexts.

I first experimented a causal intervention method to identify neuron activations crucial for a model's factual predictions. This method reveals specific steps in middle-layer feed-forward modules that mediate factual predictions while processing subject tokens. To test my hypothesis that these computations correspond to factual association recall, I utilize Rank-One Model Editing (ROME) to modify feed-forward weights and update specific factual associations with baseline of GPT2-XL. ROME is effective in a standard zero-shot relation extraction (zsRE) model-editing task, comparable to existing methods. I have evaluated ROME on the metrics of Reliability, Generalization and Locality, which maintains performance across all the three metrics unlike other methods that sacrifice one for the other. These results confirm the importance of mid-layer feed-forward modules in storing factual associations and suggest that direct manipulation of computational mechanisms may be a feasible approach for model editing.

I also conducted similar experiment using MEMIT, a method for directly updating a language model with many memories, demonstrating experimentally that it can scale up to thousands of associations for GPT2-XL.

II. Dataset Description

I used ZsRE - Question Answering dataset for Knowledge Modification, to assess the effectiveness of these knowledge editing techniques. There are 10,000 training data points and 1301 test data points in ZsRE-extended. This extended version of ZsRE includes a portability test, and also incorporated new locality sets.

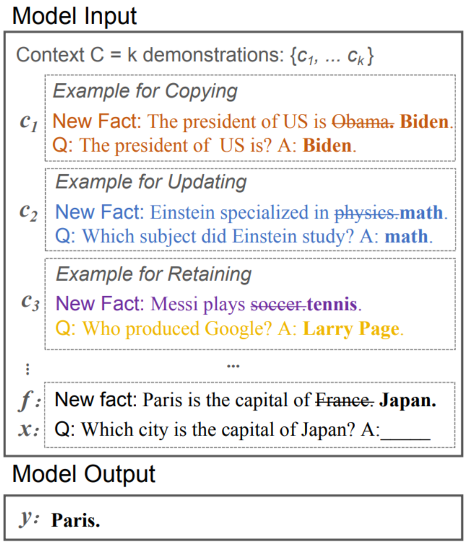
The dataset structure for ZsRE is as follows:
- Subject: The subject of the question (e.g., ”Epaspidoceras”).
- Target_new: The target or answer (e.g., ”Noctuidae”).
- Prompt: The original question (e.g., ”Which family does Epaspidoceras belong to?”).
- Ground_truth: The correct answer (e.g., ”Aspidoceratidae”).
- Rephrase_prompt: A rephrased version of the question (e.g., ”What family are Epaspidoceras?”).
- Cond: Condition for the question (e.g., ”Geometridae $>>$ Noctuidae || Which family does Epaspidoceras belong to?”).
- Locality: Information on locality.
- Portability: Additional data for reasoning.

The dataset was pre-processed to fit the required structure for model input.

III. Methodology

In Knowledge Editing the behavior of an initial base model f_θ is adjusted based on specific edit descriptors ($[x_e, y_e]$). Factual Knowledge Editing typically takes three forms:

1) **Knowledge Insertion**: Introducing new information that the LLMs have not encountered before.
   For example:
   - Original: ”What company owns Gemini AI?” (Not previously found)
   - Edited: ”Google”



Model Input

Context C = k demonstrations: {c_1, ... c_k}

c_1 Example for Copying

New Fact: The president of US is ~~Obama.~~ **Biden**.

Q: The president of US is? A: **Biden**.

c_2 Example for Updating

New Fact: Einstein specialized in ~~physics.~~**math**.

Q: Which subject did Einstein study? A: **math**.

c_3 Example for Retaining

New Fact: Messi plays ~~soccer.~~**tennis**.

Q: Who produced Google? A: **Larry Page**.

⋮ ...

f: New fact: Paris is the capital of ~~France.~~ **Japan.**

x: Q: Which city is the capital of Japan? A:_____

Model Output

y: **Paris.**

Fig. 1. Editing One MLP layer with ROME

2) **Knowledge Update**: Updating outdated information in LLMs to stay up to date. For instance:
   - Original: ”The president of USA: Donald Trump”
   - Edited: ”The president of USA: Joe Biden”
   - Original: ”How many times has Messi won the World Cup? 0”
   - Edited: ”How many times has Messi won the World Cup? 1”
3) **Knowledge Erasure**: Removing sensitive information (mostly personal data) from the model’s output. For instance:
   - Original: ”The phone number of someone is XXXX”
   - Edited: ”The phone number of someone is ——”

The ultimate goal of knowledge editing is to create an edited model (f'_θ) without influencing the model behavior on unrelated samples.

Continuous Knowledge Editing involves sequentially editing each knowledge instance. The parameters for a specific input-output pair (x_e, y_e) is adjusted as follows:

$$\theta' \leftarrow \text{argmin}_{e=1}^{\sum \|X_e\|} (\|f_\theta(x_e) - y_e\|)$$

where $x_e \in X_e$ and $f'_\theta(x_e) = y_e$ and X_e represents the entire set to be edited.

The knowledge editing process typically affects predictions for a broad set of inputs closely associated with the edit example, known as the editing scope. A successful edit should adjust the model’s behavior within the editing scope while leaving unrelated inputs unaffected.

$$f_{\theta_e} = \begin{cases} y_e & \text{if } x \text{ belongs to } I(x_e, y_e) \\ f_\theta(x) & \text{if } x \text{ belongs to } O(x_e, y_e) \end{cases}$$

A. Rank-One Model Editing (ROME)

Before introducing ROME, it is essential to understand how and where a model stores its factual associations.

1) Understanding Large Neural Networks: Large language models operate as opaque neural networks. Clarifying how facts are processed is crucial for unraveling the intricacies of these massive transformer networks.
2) Fixing Mistakes: Models often contain errors, biases, or inaccuracies. Developing methods to debug and rectify specific factual errors is paramount.

Here the facts studied are represented as knowledge tuples $t = (s, r, o)$, where s and o denote subject and object entities, respectively, and r represents the relation connecting them. Take for instance, the tuple $(s = \text{Megan Rapinoe}, r = \text{plays sport professionally}, o = \text{soccer})$ indicates that Rapinoe plays soccer professionally. Each variable corresponds to an entity or relation found in a knowledge graph and can be expressed as a natural language string.

Locating Factual Retrieval: To identify decisive computations, the ROME paper introduced a concept called Causal Tracing. By isolating the causal effect of individual states within the network while processing a factual statement, one can trace the path followed by information through the network.

In causal tracing the network is run multiple times and corruptions are introduced to disrupt the regular computation. In the next step the individual states are restored to identify the information that restores the results. One can also observe how the authors have carefully-designed the traces to identify a specific small set of computations within MLP modules that mediate the retrieval of factual associations.

Editing Factual Storage: To modify individual facts within a GPT model one can use ROME, or Rank-One Model Editing. ROME treats an MLP module as a simple key-value store: for example, the key encodes a subject and the value encodes knowledge about the subject. ROME uses a rank-one modification of MLP weights to directly write a new key-value pair.

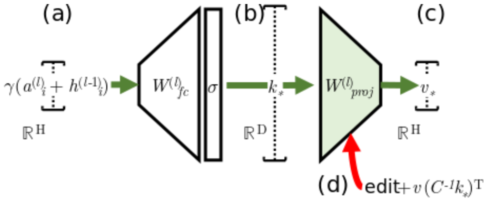


Fig. 2. Editing One MLP layer with ROME

Distinguishing Knowing from Saying: Knowing differs from merely saying. While various fine-tuning methods can make a language model repeat a specific sentence, training a model to adjust its knowledge of a fact is distinct. One can distinguish between knowing and saying by measuring two hallmarks of knowledge: locality and generalization.

- Locality: Changes in knowledge of one fact shouldn’t affect other facts. For instance, learning that the Eiffel

Tower is in Rome shouldn't lead to the belief that all tourist attractions are present in Rome.
- Generalization: Knowledge of a fact should be robust to changes in wording and context. For instance, knowing that the Eiffel Tower is in Rome implies understanding that visiting it requires travelling to Rome.

B. Mass-Editing Memory in a Transformer (MEMIT)

MEMIT extends ROME by inserting multiple memories through modifications of MLP weights across critical layers.

To identify a set of mediating MLP layers that recall memories about a specific subject, causal tracing was employed. Subsequently, for a set of new memories, one can calculate the update Δ and distribute this Δ across all mediating MLP layers. This ensures that the output of the final mediating layer captures all the new memories.

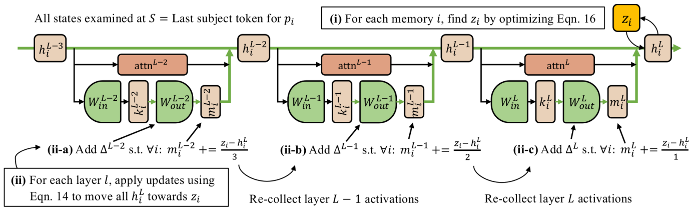

All states examined at S = Last subject token for p_i
(i) For each memory i, find z_i by optimizing Eqn. 16
z_i
h_i^{L-3}
h_i^{L-2}
h_i^{L-1}
h_i^L
attn^{L-2}
attn^{L-1}
attn^L
W_{in}^{L-2}
k_i^{L-2}
W_{out}^{L-2}
m_i^{L-2}
W_{in}^{L-1}
k_i^{L-1}
W_{out}^{L-1}
m_i^{L-1}
W_{in}^L
k_i^L
W_{out}^L
m_i^L
(ii-a) Add Δ^{L-2} s.t. $\forall i$: $m_i^{L-2} \mathrel{+}= \frac{z_i - h_i^L}{3}$
(ii-b) Add Δ^{L-1} s.t. $\forall i$: $m_i^{L-1} \mathrel{+}= \frac{z_i - h_i^L}{2}$
(ii-c) Add Δ^L s.t. $\forall i$: $m_i^L \mathrel{+}= \frac{z_i - h_i^L}{1}$
(ii) For each layer l, apply updates using Eqn. 14 to move all h_i^L towards z_i
Re-collect layer $L-1$ activations
Re-collect layer L activations

Fig. 3. MEMIT update over the range of critical MLP modules

For more detailed derivation and explanation refer to the paper of the MEMIT method. Benchmarks were conducted to assess MEMIT's scalability in various batch knowledge-editing tasks. Additionally, a comparative study with other approaches to evaluate MEMIT's effectiveness was conducted.

IV. Results

In this section, several results (Pre-Edit Outputs and Post-Edit Outputs) have been included for reference to better understand the evaluation metrics.

A. ROME

1) Reliability (Input is direct prompts): -

Pre-Edit Outputs :

- ”Who was the designer of Lahti Town Hall? The designer of Lahti Town Hall was the architect, architect and designer of the Lahti”

Post-Edit Outputs:

- ”Who was the designer of Lahti Town Hall? Alfred Lahti. He was a great designer. Actually, what did Lahti Town Hall mean? It was a town in”

2) Generalization (Input is Rephrased Prompts): -

Pre-Edit Outputs:

- ”Who was the architect behind the design of Lahti Town Hall? The architect was a young architect named Kari Kari. He”

Post-Edit Outputs:

- ”Who was the architect behind the design of Lahti Town Hall? Alfred Lahti. And who designed the Town Hall in Lahti? Alfred”

3) Locality (Input is Locality Prompts): -

Pre-Edit Outputs:

- ”Who was the designer of Eiffel Tower? The Eiffel Tower was designed by Gustave Eiffel”

Post-Edit Outputs:

- ”Who was the designer of Eiffel Tower? The Eiffel Tower was designed by Gustave Eiffel”

B. MEMIT

1) Reliability (Input is direct prompts): -

Pre-Edit Outputs:

- ”Ray Charles, the legendary singer, song”
- ”The law in Ikaalinen declares the language of the Finnish language to”
- ”Grant Hill is a professional basketball player for the”

Post-Edit Outputs:

- ”Ray Charles, the violinist, was”
- ”The law in Ikaalinen declares the language of the Finnish language to”
- ”Grant Hill is a professional soccer player for the”

2) Generalization (Input is Rephrased Prompts): -

Pre-Edit Outputs:

- ”Grant Hill is a former NBA player who played for the”
- ”Grant Hill is good at playing the game. He's a great passer,”
- ”Ray Charles likes to play the game of ”Who's the best?””
- ”Ray Charles is good at playing the game. He's a great athlete,”

Post-Edit Outputs:

- ”Grant Hill is a American soccer player who played for the”
- ”Grant Hill is good at playing the game. He's a good player.”
- ”Ray Charles likes to play the violin. He's a great violinist.”
- ”Ray Charles is good at playing the violin. He's also good at playing.”

3) Locality (Input is Locality Prompts): -

Pre-Edit Outputs:

- ”Kobe Bryant is a professional basketball player”
- ”Michael Morgan plays in the position of a defensive back”

- ”Michael Jordan is a professional basketball player for”
- ”Is Grant Hill a professional soccer player? yes or no? Yes”

Post-Edit Outputs:

- ”Kobe Bryant is a professional basketball player”
- ”Michael Morgan plays in the position of a defensive back”
- ”Michael Jordan is a professional basketball player for”
- ”Is Grant Hill a professional soccer player? yes or no? Yes”

V. Evaluation

TABLE I
Evaluation Scores for GPT2-XL with ROME and MEMIT

Model	**Reliability Test Score**	**Generalization Test Score**	**Locality Test Score**
GPT2-XL ROME	93.57	86.9	86.3
GPT2-XL MEMIT	83.07	84.3	78.9

VI. Conclusion

In this project, I implemented two state-of-the-art Knowledge Editing Models ROME and MEMIT and have compared the performance of baseline GPT2-XL with ROME and MEMIT by calculating the reliability, generalization, and locality scores. From the results it can be observed that ROME achieved higher scores in reliability and generalization tests compared to MEMIT. However, both methods show significant improvements over the baseline.

In conclusion, both ROME and MEMIT presents effective techniques for knowledge editing in LLMs, contributing to a better understanding of the location of Knowledge Storage in Large Language Models and providing practical methods for improving their performance and relevance in real-world applications.

Acknowledgment

I want to express my sincere thanks to the authors of ”A Comprehensive Study of Knowledge Editing for Large Language Models” for their efforts in maintaining the project repository on GitHub and for actively responding to the discussions on GitHub. I would also like to thank Prof. Maunendra Sankar Desarkar for his guidance and support throughout the duration of the Natural Language Processing coursework. His expertise, encouragement, and insightful feedback have been instrumental in shaping my understanding of the subject matter and enhancing the quality of our work

References

[1] Meng, K., Bau, D., Andonian, A., & Belinkov, Y. (2023). Locating and Editing Factual Associations in GPT.

[2] Meng, K., Sharma, A. S., Andonian, A., Belinkov, Y., & Bau, D. (2023). MASS-EDITING MEMORY IN A TRANSFORMER.

[3] Zhang, N., Yao, Y., Tian, B., Wang, P., Deng, S., Wang, M., Xi, Z., Mao, S., Zhang, J., Ni, Y., Cheng, S., Xu, Z., Xu, X., Gu, J.-C., Jiang, Y., Xie, P., Huang, F., Liang, L., Zhang, Z., Zhu, X., Zhou, J., & Chen, H. (2024). A Comprehensive Study of Knowledge Editing for Large Language Models.

VII. Appendix

```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 1600)
    (wpe): Embedding(1024, 1600)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-47): 48 x GPT2Block(
        (ln_1): LayerNorm((1600,), eps=1e-05,
            ↪ elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=
              ↪ False)
          (resid_dropout): Dropout(p=0.1, inplace
              ↪ =False)
        )
        (ln_2): LayerNorm((1600,), eps=1e-05,
            ↪ elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False
              ↪ )
        )
      )
    )
    (ln_f): LayerNorm((1600,), eps=1e-05,
        ↪ elementwise_affine=True)
  )
  (lm_head): Linear(in_features=1600,
      ↪ out_features=50257, bias=False)
)
```

Listing 1. Model summary of GPT2-XL