Twice is Nice!
Improving MEMIT for Knowledge Editing

Anonymous ACL submission

Abstract

In this ever changing world, Large Language Models (LLMs) pose a major challenge on frequently retraining as they consume enormous resources. Direct knowledge editing emerged as an efficient alternative, where it locates a stale factual knowledge in the LLM's layers and edits those layers' weights in order for the LLM to generate new factual knowledge. We observed that MEMIT (Meng et al., 2023b), state-of-the art knowledge editing algorithm is not used at its full potential. In this paper, we empirically demonstrated the limitations of executing only one single MEMIT update. We then proposed an intuitive and straightforward solution, Running MEMIT twice, and showed its effectiveness over two knowledge editing datasets compared to strong baselines. We conducted extensive analysis to understand the effectiveness of our solution. In particular we analyzed multiple runs of MEMIT and found out the performance to plateaus at second run of MEMIT. To discern the reason we analyzed the gradients between each run and found negligible change in gradients between second and third run of MEMIT.

1 Introduction

Large language models (LLMs) have been widely used as source of knowledge (Radford et al., 2019a; Brown et al., 2020; Wang and Komatsuzaki, 2021; Black et al., 2022; Petroni et al., 2020) to retrieve factual or specialized knowledge. However, knowledge can change, facts can become outdated. For example, Twitter re-branded to 'X'. So, it's desirable in many application domains like content generation, question answering and knowledge retrieval to have flexible models capable to adapting to these changes. Since re-training an LLM consumes a lot of resources (Patterson et al., 2021), knowledge editing has emerged as an efficient alternative to update the stale knowledge (Lazaridou et al., 2021; Agarwal and Nenkova, 2022; Liška et al., 2022).

MEMIT (Meng et al., 2023b) stands as the one of the most efficient knowledge editing technique in literature (Meng et al., 2023a; Zhu et al., 2020; Cao et al., 2021; Hase et al., 2021; Mitchell et al., 2022a,b). It can edit effectively thousands of knowledge instances simultaneously in an LLM. Given a factual knowledge tuple (s, r, o) with subject s, relation r, and object o, MEMIT views the subject s as key and the object o as a value under the relation r. MEMIT modifies the transformer weights such that the edited model generates new object o^* instead of o given the (s, r) pair. Figure 1 shows an example.

MEMIT begins by identifying the relevant layers (the green layers in Figure 1) that contribute the most to the knowledge tuple (s, r, o) using causal tracing (Pearl, 2013; Vig et al., 2020; Meng et al., 2023a). Then, the update process consists of two stages. In Stage 1, the model learns a hidden vector δ denoting the change at the final relevant layer L through a standardized optimization method, in order to generate the new token o^* from the final output layer. In Stage 2, the updated final relevant layer L is spread across all relevant layers such that each of them contributes roughly equally to the final prediction.

While an advance over existing methods, MEMIT still makes many errors in practice. We hypothesize that the failed cases may be attributed to the optimization process in Stage 1. Although the last relevant layer L is one of the most important layers for making the final prediction identified by casual tracing, all parameters in the relevant layers contribute significantly to the prediction, too. When MEMIT only optimizes vector δ at layer L, all other layer parameters remain unchanged, thus limiting the final loss landscape in predicting the new token.

To validate our hypothesis, we pose the follow-

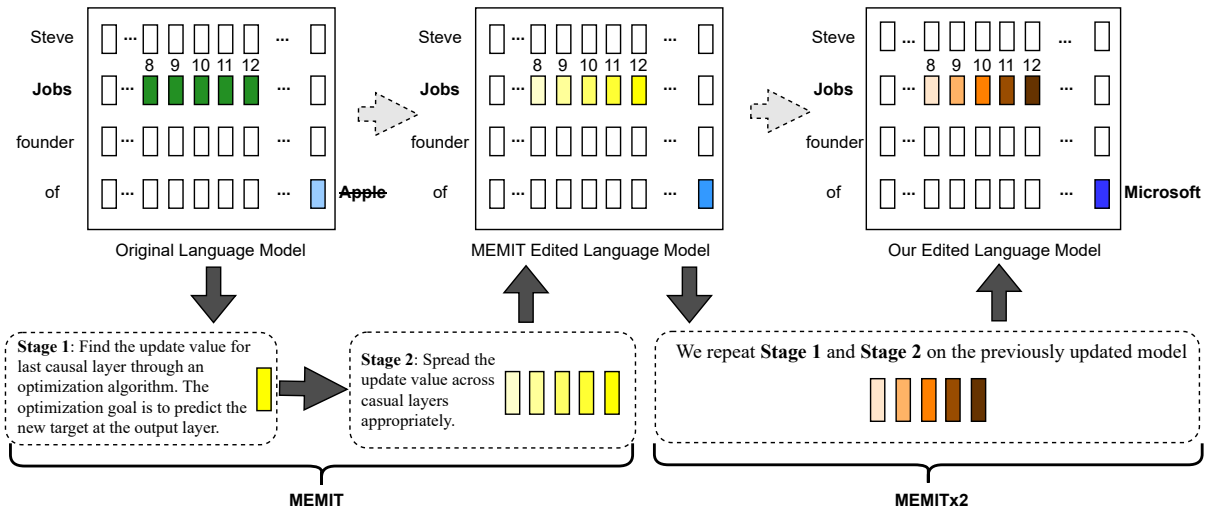

Figure 1: Stages of MEMIT and MEMIT×2. The goal is to make the model generate 'Microsoft' instead of 'Apple' given the prompt - 'Steve Jobs founder of.' Causal layers (indicated in green) are responsible for recalling facts and are identified through causal tracing. MEMIT finds the updated value for the last causal layer and then spreads it across all layers (shown with shades of yellow). MEMITx2 repeats this process again.

ing research questions. **RQ1**: Is it true that the optimization failure is due to constrained loss landscape and that it is not because of an insufficient training of the optimization algorithm? **RQ2**: What if we update the other relevant layers and then utilize the optimization algorithm to find a new update value, i.e run MEMIT twice? **RQ3**: How many time do we need the updates? Is it true that, the more, the better? To address these questions, we performed extensive experiments and analysis on a *hard* knowledge editing subset, i.e MEMIT failed to edit these examples.

We first show that extending the optimization steps to find δ does not result in edit success but sometimes leads to diminished model performance. We then illustrate that simply running MEMIT again, i.e., updating model parameters in the other relevant layers followed by another optimization step to find the new δ value, yields significant gain. We also illustrated that twice is enough; the performance plateaus after the second update, indicating the importance of the relevant layers. Only updating the relevant layers without updating any other layers yields almost 100% edit successes. Additionally, we intuitively demonstrate the update gradient difference between MEMIT, MEMIT×2, and MEMIT×3. This confirms the importance of the second update in guiding δ in a different but correct direction. Finally, we report a more comprehensive results table using two models on two datasets, and show significant performance improvement when running MEMIT twice as opposed to just once.

2 Background

MEMIT performs edits on a given model. Specifically, the method takes an input prompt x consisting of the concatenation of the subject s and verb v and directly edits the model parameters such that the model predicts the new object o^* instead of the previous object o as the prediction y.

To achieve this, MEMIT identifies the model layers responsible for knowledge prediction using causal tracing (Pearl, 2013; Vig et al., 2020; Meng et al., 2023a). These layers are viewed as the critical layers in predicting the object token given s and v. We briefly describe the intuition of casual tracing, please refer to Meng et al. (2023b) for more details. If the model input is noisy, the output should also be noisy. The substitution of any layer value in the noise network produces the original output; the layer plays a crucial role, and the layers are considered responsible for the original output. MEMIT selects the layer L with the most significant impact on the final output layer and includes its four preceding layers as the critical causal layer, $L-1$ to $L-4$.

MEMIT then performs parameter updates on these layers. The update process includes two stages. In the first stage, the method aims to find the update needed for the final casual layer L, represented by z. The calculation of z is shown below:

$$z = h^L + \text{argmin}_\delta \frac{1}{N} \sum_{i=1}^{N} -\log \mathbb{P}(o^* | x_i \oplus (s, r))$$

where h^L is the hidden value of layer L, δ is optimized to generate the desired object o^* given N prompts, the prompts are the concatenation of random factual prefixes x_i and the knowledge prompt s and r. The optimization is done via Adam (Kingma and Ba, 2014).

In the second stage, MEMIT spreads the learned z vector across all causal layers. The method assumes the last causal layer contributes the most, and the contribution decreases as we go to the preceding layers. The spread is done proportionately relative to the distance from the layer L. This implies that the furthest causal layer gets the lowest weight update. We follow the exact update equations as MEMIT (Meng et al., 2023b).

3 Experiments

The two-stage update in MEMIT has its limitations, resulting in imperfect editing outcomes. We hypothesize that the one of the main causes of the failed cases is attributed to the optimization process in the first stage. A typical optimization failure could be a lack of training time. We thus investigated whether the update was terminated prematurely. To address **RQ1**, we design our experiments by adjusting MEMIT's final step to various values. We expect to see an improvement in edit success if this is the cause of the failed cases. If running time is not the cause of the failed case, this indicates that gradient descent algorithms were unable to find the optimal σ that results in the new object o^* in the final layer. Our second **RQ2** studies if changing the optimization loss landscape by updating the other model parameters helps. To answer **RQ2**, we made two changes to the typical MEMIT running. Firstly, instead of executing MEMIT a single time, we ran it twice with two updates. Secondly, we maintained the number of gradient steps consistent with the default value (i.e., 20) during the second update. This aims to study the update effects independent of the number of gradient steps used.

Datasets Similar to Meng et al. (2023b), we performed our experiments on COUNTERFACT (Meng et al., 2023a) and zsRE (Levy et al., 2017) datasets. Both consists of ~20k factual knowledge. In particular, COUNTERFACT consists of factual statements which are converted to *counterfactual*, whereas zsRE is an question-answering dataset from which real-world facts are extracted (please refer to Meng et al. (2023b) for more details). So, the former test's MEMIT's ability to add *counterfactual* information and the later test's the ability to add *correct* information.

LLMs We run our experiments on three auto regressive LLMs: GPT-2 L (774M parameter; Radford et al. (2019b)), GPT-2 XL (1.5B parameters; Radford et al. (2019b)) and GPT-J (6B parameters; Wang and Komatsuzaki (2021)). We used the same hyper-parameters and causal layers reported by Meng et al. (2023b). Although, during certain experiments we varied the gradient steps, discussed in later sections. See Appendix A for resource and computation details.

Evaluation We used the same evaluation metrics as MEMIT - Efficacy, Generalization, and Specificity. Efficacy measures the success of knowledge rewrite, i.e., the edited model should output "Microsoft" for the example in Figure 1. Generalization measures if the edit is superficial and overfitted by testing the edited model on paraphrases where the output should also be changed. On the other hand, specificity measures the neighboring knowledge that should not be changed.

For each of these three metrics, we report two evaluation numbers – success and accuracy. To be specific, success gauges the relative score between the new target o^* and the previous target o. Efficacy success means the score of o^* surpasses that of o. Conversely, specificity success is true when the reverse is observed. Accuracy, however, is a more stringent measure. Accuracy is true when o^* emerges as the top-1 prediction given (s, r) for both efficacy and generalization, while for specificity, o remains the top-1 generation. To provide a comprehensive evaluation, we report the harmonic mean of the three metrics: efficacy, generalization, and specificity, thereby encapsulating all relevant information into a singular score, resulting in SUCCESS SCORE and ACCURACY SCORE.

4 RQ1: More gradient descent steps does not help

To measure the significance of gradient steps (used in stage-1) in MEMIT's performance we evaluated the edited model at multiple step values, by increasing the steps by 5 from 20 (default setting in

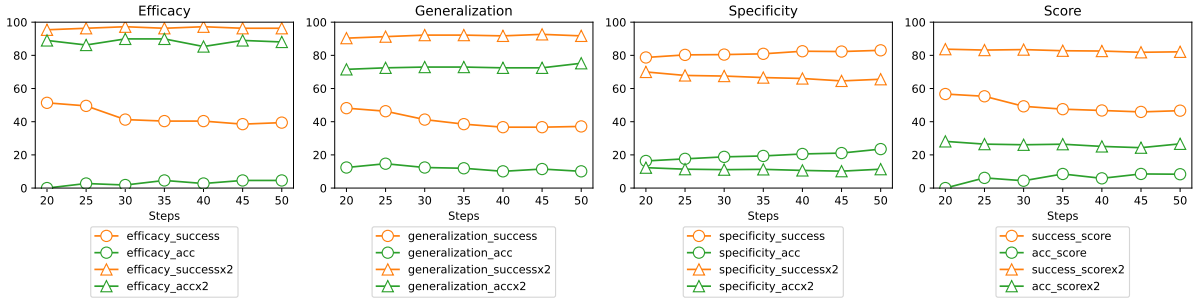

Figure 2: MEMIT vs MEMIT×2. We aim to examine the significance of optimization steps in MEMIT's stage 1. We plot the evaluation results on HARD EDIT SET for gradient steps from 20 to 50. Circle (○) lines represent MEMIT. We also aim to compare MEMIT and MEMIT×2, so we also plot the evaluations results for MEMIT×2 on HARD EDIT SET, represented with triangle (Δ) line.

MEMIT) to 50[1].

4.1 Experiment

We perform this experiment on COUNTERFACT dataset with GPT-2 XL. We choose the causal layers for GPT-2 XL reported by Meng et al. (2023b) to perform the edit. We randomly sample 1000 examples from this dataset due to the time complexity in running these experiments, denoted as SAMPLE SET.

We first run and evaluate MEMIT on SAMPLE SET with default number of gradient steps (20). This results in a collections of examples for which MEMIT failed to edit (evaluated by EFFICACY ACCURACY). We report the evaluation results (should be 0) on this subset. We call this a HARD EDIT SET. By extracting this subset, we are able to magnify the performance changes on these hard examples. Then we run and evaluate MEMIT on an un-edited model with SAMPLE SET again, however, increase the number of gradient steps by 5. Same as before, we report results on HARD EDIT SET. We gradually increase the gradient steps on un-edited models until it reached 50.

We acknowledged that the failed instances may vary between different MEMIT runs. However, the set of failed instances for lower gradient steps is likely to include examples with more gradient updates. So we report results on HARD EDIT SET extracted from the first run.

4.2 Analysis

The results are illustrated as circle (○) lines in Figure 2. We delineate the pattern of efficacy, generalization, specification, and average success/accuracy measure as the number of gradient steps increased. The overall success/accuracy revealed that increasing the number of gradient steps adversely affects MEMIT's performance, especially SUCCESS SCORE.

Looking at the individual metrics, both efficacy and generalization performance show a downward trend. The EFFICACY ACCURACY starts at 0% by experimental design. It increases marginally at 25 gradient steps then starts decreasing and remains stagnant around lower single-digit % values. The EFFICACY SUCCESS and GENERALIZATION SUCCESS showed a much bigger drop. Both metrics initiated around 50%, followed by a much bigger performance drop after a small increase at 25 steps. This could happen when the number of examples that achieved success but not accuracy at gradient step 20 decreases. Further optimization of δ results in a lower probability of o^* compared to o. We think that the decline in performance is attributable to the gradient deviating from the minima, indicating that MEMIT did not terminate early.

The specificity performance shows a slight increase in success. This could be caused by the "undercorrecting" behavior of the model. In instances where the model is unable to effectively edit the original example, it tends to retain the same prediction for neighboring knowledge elements due to the similarity in their representations. However, when assessing the overall performance, it is evident that increasing the number of steps does not positively impact the HARD EDIT SET.

5 RQ2: Run MEMIT again is effective

The optimization problem of finding the updated value of δ for the final relevant layer L to maximize the probability of the target o^* is in stage 1

[1]We did not see much performance change so we stopped at step 50.

Model	GPT 2 XL			GPT 2 L			GPT J		
Metric	**ROME**	**MEMIT**	**MEMIT×2**	**ROME**	**MEMIT**	**MEMIT×2**	**ROME**	**MEMIT**	**MEMIT×2**
Efficacy success	66.8	94.2	**99.5 (+5.3)**	49.7	86.3	**99.3 (+13)**	55.2	98.7	**99.9 (+1.2)**
Efficacy accuracy	31.0	88.3	**98.1 (+9.8)**	0.4	78.7	**97.6 (+18.9)**	6.8	97.9	**99.1 (+1.2)**
Generalization success	64.75	88.35	**97.55 (+9.2)**	49.4	78.25	**96.85 (+18.6)**	53.65	92.15	**98.45 (+6.3)**
Generalization accuracy	27.2	72.95	**90.3 (+17.35)**	0.45	60.85	**88.8 (+27.95)**	8.4	83.4	**92.0 (+8.6)**
Specificity success	51.33	**63.56**	59.4 (-4.16)	50.29	**70.41**	60.89 (-9.52)	55.0	**74.74**	66.38 (-8.36)
Specificity accuracy	8.81	**17.21**	14.37 (-2.84)	0.22	**21.44**	14.32 (-7.12)	2.66	**27.0**	23.2 (-3.8)
Score success	60.12	79.64	**80.78 (+1.14)**	49.79	77.78	**81.48 (+3.7)**	54.60	**87.30**	85.14 (-2.16)
Score accuracy	16.43	**36.08**	33.01 (-3.07)	0.32	**39.58**	32.84 (-6.7)	4.67	**50.63**	46.82 (-3.81)

Table 1: Result of multiple runs of MEMIT on CounterFact (1000 Examples)

Model	GPT 2 XL			GPT 2 L			GPT J		
Metric	**ROME**	**MEMIT**	**MEMIT×2**	**ROME**	**MEMIT**	**MEMIT×2**	**ROME**	**MEMIT**	**MEMIT×2**
Efficacy acc	61.11	79.6	**98.59 (+18.99)**	77.58	66.11	**96.02 (+29.91)**	88.05	99.4	**99.78 (+0.38)**
Generalization acc	54.01	72.96	**94.03 (+21.07)**	76.6	60.37	**91.26 (+30.89)**	83.64	94.37	**97.56 (+3.19)**
Specificity acc	22.47	**25.96**	25.2 (-0.76)	18.75	23.89	**24.25 (+0.36)**	24.5	**28.32**	28.13 (-0.19)
Score acc	37.79	46.30	**49.61 (+3.31)**	37.84	40.78	**47.91 (+7.31)**	46.78	53.60	**53.74 (+0.14)**

Table 2: Result of multiple runs of MEMIT on zsRE (1000 Examples). Similar to Meng et al. (2023b) we report accuracy values.

of MEMIT. The learning takes place when all the relevant layers are fixed in the model. However, all relevant layers should make a significant contribution to the final prediction according to causal tracing. This motivates us to spread the changes across layers (stage 2 of MEMIT) and then find the optimal value δ^* again (stage 1 of MEMIT), as depicted in Figure 1. This is essentially, running MEMIT twice, i.e two updates.

5.1 Experiment

We follow a similar experimental setup outlined in subsection 4.1 with one modification. The modification is that we ran MEMIT for two updates instead of one. To achieve this, the updated weights learned from the first update were saved and used as the initial parameters for the second update. To study the implications of conducting two MEMIT updates, we only vary the number of gradient steps during the first update (same as subsection 4.1), while the number of gradient steps in the second update was fixed at the default value of 20.

5.2 Analysis

Figure 2 shows the results of MEMIT×2 as lines with triangle marker. When comparing MEMIT and MEMIT×2, a notable observation is that both the success score and accuracy score of MEMIT×2 significantly surpass those of MEMIT, regardless of the gradient update steps. Furthermore, some individual categories reveal a comparable improvement, specifically, in efficacy and generalization for both success and accuracy measures. However, a drop is observed in the specificity performance. We hypothesize that MEMIT×2 "overcorrected" the output. Due to the representation similarity between neighbors, the model generates the new object o^* instead of the original object o even for the neighbor prompts. Nonetheless, the gains achieved in other metrics markedly outweigh the drop in specificity, leading to an overall score improvement in comparison to MEMIT.

The second observation is that the number of gradient steps is irrelevant in terms of the overall score for MEMIT×2, the overall score remains relatively flat. The same is true for the individual categories. It was interesting to observe the change in relationships between the number of gradient steps, with MEMIT and MEMIT×2. The Success Score decreased as the gradient step increased in MEMIT, while Success Score remains unchanged in MEMIT×2. This observation also suggests that spreading the updates from stage 1 facilitates the optimization problem of finding the updated value for the prediction of the new object.

5.3 Additional Quantative Results

To strengthen our findings and generalize the results we evaluated MEMIT and MEMIT×2 on two benchmark datasets CounterFact and zsRE and used GPT-2 L, GTP-2 XL and GPT-J LLMs. We also compared our results with strong baseline in-

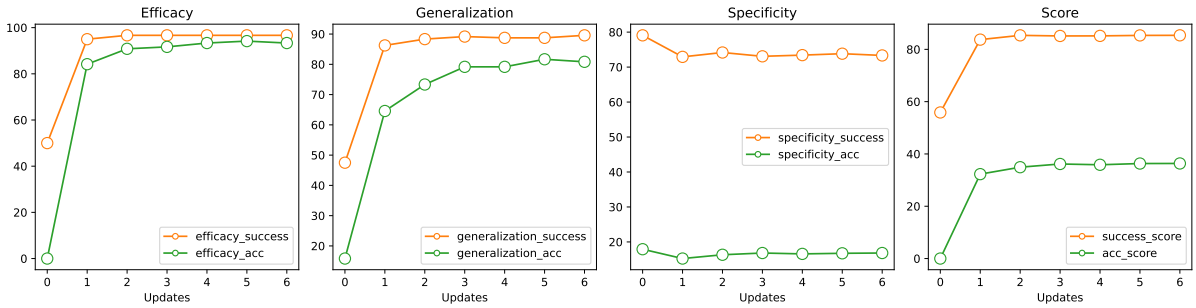

Figure 3: Performance trend through multiple updates of MEMIT

cluding ROME (Meng et al., 2023a) and MEMIT. The results for COUNTERFACT are shown in Table 1 and for zsRE in Table 2.

It can be observed from Table 1 and Table 2 that there's a significant increase in efficacy and generalization for both success and accuracy measures. The gain for COUNTERFACT and zsRE in percentage points for both GPT-2 L and GPT-2 XL is huge ranging from 5 to as high as 30 points.

As expected, we observed a drop in specificity across the board, with GPT-2 L on zsRE being an exception (see Table 2). The drop in percentage points is less compared to gains achieved in efficacy and generalization. This drop didn't negatively effect SUCCESS SCORE and ACCURACY SCORE for COUNTERFACT and zsRE, respectively. However, do negatively effect on ACCURACY SCORE for COUNTERFACT.

In case of GPT J, the gain is not as significant as compared to other two models because the margin for improvement is very small in GPT J from MEMIT to MEMIT$\times$2. Due to this lower improvement margin, there's more negative effect on the overall score for the COUNTERFACT.

To summarize, MEMIT$\times$2 give significant boost to efficacy and generalization performance across success and accuracy measures. This proves our hypothesis that re-optimization (stage 1) after spreading initial δ to relevant layers (stage 2) will result in better loss landscape, in turn finding a better δ and improving MEMIT's performance. However specificity do take a slight hit which we hypothesize is due to "overcorrected" output.

6 RQ3: Performance plateaus with multiple MEMIT runs

Since, our analysis of **RQ2** made it clear that running MEMIT for two updates improves the performance, The next question is how many updates are required? What is the relationship between the number of updates and the performance? To address **RQ3** we ran MEMIT for multiple updates.

6.1 MEMIT$\times n$

Experiments To accomplish this goal, we conducted an experiment by frequently updating the model through multiple iterations of MEMIT. We reduced the number of gradient steps in stage 1 allowing us to run stage 2 more frequently. With this experiment we want to analyse the chance in MEMIT's performance wrt to number of updates. To conduct this experiment, we used the same experimental setup described in subsection 4.1, i.e all results are reported on HARD EDIT SET, and edits are done on SAMPLE SET. However, instead of starting from an unedited model at every step interval we started with an edited model of the previous step interval. So, after every 5 steps of gradient descent in stage 1 the model is updated through parameter spreading, i.e stage 2. Update n always displays results of a MEMIT run with five gradient steps in the first stage, and this MEMIT run is editing the model from update $n-1$.

It is important to emphasize that this edited model at "Update 1" (Fig. 3) is different from that used at step 25 (Fig. 2) while addressing **RQ1**. "Update 1" edited on the "Update 0" model, while step 25 edited on the original model.

Analysis Fig. 3 shows the experiment results, looking at SUCCESS SCORE and ACCURACY SCORE, there is a huge jump from update 0 to update 1, indicating the effectiveness of MEMIT$\times$2, the performance starts to platues after update 1. Looking closer, we saw a continuous subtle improvements in ACCURACY SCORE, EFFICACY ACCURACY, and GENERALIZATION ACCURACY. Recall that it's easier to achieve success before accuracy as the former requires the LLM probability of

new object $(o*)$ to be greater than the old object (o). Whereas in later the same probability has to be maximum across the vocabulary given (s, r), which is harder to achieve than the former. So intuitively, model reached success bar, and continues to increase the desired object project, resulting in a slow increase in accuracy.

In case of specificity, as expected, we saw a reverse trend compared to the other two metrics, as show in Fig. 3. We observed sharp decrease for first update then slight improvement in the second a then a plateau for the rest. Both success and accuracy metric followed the same trend. An interesting thing to notice was that the decrease in specificity performance is way less compared to the increase in efficacy and generalization performance. So with small sacrifice of specificity performance, we achieve a bigger boost in the efficacy and generalization performance (shown in Table 1 and Table 2).

These results in general shows the strong effectiveness of MEMIT×2, and limited effectiveness of $\times n$ where $n > 2$.

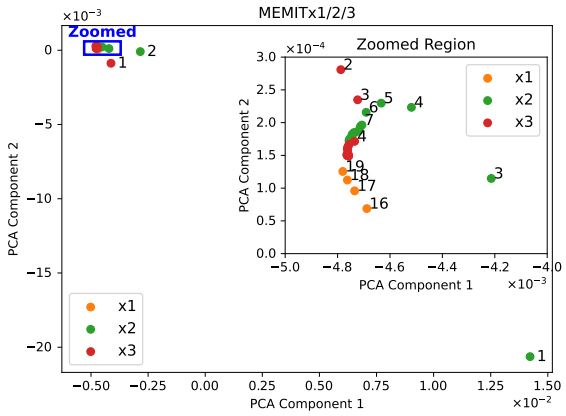

Figure 4: Gradient Analysis. We aim to analyze the difference between the gradients of multiple updates of MEMIT. We plot 2 principle components for last four gradients of 1st and all gradients of 2nd and 3rd update. Since the gradients are very close (blue region) we also plot its zoomed view. The first gradient of MEMIT×2 is very different from all the other points indicating its importance in finding the optimal value.

6.2 PCA analysis of the update gradient

Experiments In intuitively understand the cause of the effectiveness of MEMIT×2, we check the gradients values in steps 1 update. We hypothesize that MEMIT×2 direction of gradient is different from MEMIT but is similar to MEMIT×3. It can also be understood as the change in gradient is big between MEMIT and MEMT×2. However, the change in gradient is negligible between MEMIT×2 and MEMIT×3.

To test our hypothesis we ran MEMIT for multiple consecutive updates and compared the gradients in each update. Here each update was performed on top of previous update. In particular we ran three updates on GPT2-XL with the same random sample to 1000 examples of COUNTERFACT dataset - SAMPLE SET. Number of gradient steps was set to 20 for all three updates. During each update, we collected the gradient for all 20 steps of stage 1. We wanted to plot these gradients together for analysis. Since, the gradients were high dimensional, we applied PCA to extract out two principle components. We plot these two principle components in the 2D space as shown in Fig. 4. In particular, we plot the last four gradients for first update, represented with orange color, and all gradients for the second and third update represented with green and red color respectively.

Analysis We analyzed the relative positioning of the gradients for the three updates and found out that initial gradient for 2nd update (bottom right green point) is far from the final gradient of the 1st update (top left orange points). This shows that second gradient update provides a different direction then the end of the first one, indicating its important in finding the optimal update value. Whereas, analyzing the relative positioning of all gradients of second update (except first) and third update, we found negligible scope for optimization. This finding is evident in Fig. 3 where all green and red points are gathered in the top left area of the plot overlapping each other very close to the $(0, 0)$ co-ordinate suggesting the gradients are not changing.

This analysis proved our hypothesis that change in gradient is big between MEMIT and MEMT×2, but is negligible between MEMIT×2 and MEMIT×3. Which in turn established running MEMIT×2 as optimal because after that there is negligible scope for improvement.

7 Related Work

Large language models (LLMs) require constant updates to adapt to the ever-changing world. A straightforward approach to updating an LLM's knowledge is through fine-tuning. However, fine-tuning lacks precision, as it does not allow for spe-

cific control over the internal changes within the model, resulting in "overcorrecting" (Meng et al., 2023b).

A set of work uses a meta-learner to guide the knowledge editing. This area of study encompasses various approaches. One approach suggests a new training objective, making the model easy to edit (Sinitsin et al., 2020). Other works focus on model update values. Cao et al. (2021) use a Knowledge Editor (KE) hypernetwork to predict the necessary model updat, while MEND (Mitchell et al., 2022a), uses the gradients of inserted knowledge to predict the model update. Finally, SERAC (Mitchell et al., 2022b) adopts a gradient-free approach, caching the new knowledge in an additional memory rather than updating the model directly.

Different from all the above methods, MEMIT (Meng et al., 2023b) directly modifies the model weights to incorporate new knowledge. Similar to ROME (Meng et al., 2023a), MEMIT employs the localize-and-edit approach (described in Section 2). While ROME updates the weights of a single localized layer, MEMIT updates the weights across a range of localized layers. Furthermore, MEMIT exhibits high scalability, making it the most efficient among all prior works. Our work improves the effectiveness of MEMIT even more via a "simple trick".

8 Conclusion

In this work, we present an intuitive and straightforward approach to increase MEMIT's performance, just by ruining it twice denoted at MEMIT$\times$2. To understand the effectiveness of MEMIT$\times$2, we poses three research questions and answered them by intuitive, yet extensive analysis. Our work demonstrate the effectiveness of MEMIT$\times$2, making knowledge editing a more desirable tool for future usage.

9 Limitations

MEMIT, as discussed by Meng et al. (2023b), faces its own challenges while editing knowledge with certain relations. The experimentation is also limited to knowledge editting, however, achieves best performance in terms of Efficacy, Generalization, and Specificity with scaling to thousands of knowledge edits, compared to related works.

We propose a simple approach to improve MEMIT's performance i.e. running it twice. This solution provides a significant performance boost in terms of Efficacy and Generalization, however, it comes with a small toll on Specificity. We were not able to isolate this side effect, which opens a new research direction for further investigation. One plausible approach could be introducing an additional loss so that neighbourhood knowledge didn't change.

We were also limited by compute resources which translated to our choices of using a subset of the datasets for our experimentation. However, we have chosen an adequate size to showcase and validate our findings.

References

Oshin Agarwal and Ani Nenkova. 2022. Temporal effects on pre-trained models for language processing tasks. *Preprint*, arXiv:2111.12790.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model. *Preprint*, arXiv:2204.06745.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. *Preprint*, arXiv:2104.08164.

Peter Hase, Mona Diab, Asli Celikyilmaz, Xian Li, Zornitsa Kozareva, Veselin Stoyanov, Mohit Bansal, and Srinivasan Iyer. 2021. Do language models have beliefs? methods for detecting, updating, and visualizing model beliefs. *Preprint*, arXiv:2111.13654.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume,

Tomas Kocisky, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. 2021. Mind the gap: Assessing temporal generalization in neural language models. *Preprint*, arXiv:2102.01951.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. *CoRR*, abs/1706.04115.

Adam Liška, Tomáš Kočiský, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, Cyprien de Masson d'Autume, Tim Scholtes, Manzil Zaheer, Susannah Young, Ellen Gilsenan-McMahon, Sophia Austin, Phil Blunsom, and Angeliki Lazaridou. 2022. Streamingqa: A benchmark for adaptation to new knowledge over time in question answering models. *Preprint*, arXiv:2205.11388.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2023a. Locating and editing factual associations in gpt. *Preprint*, arXiv:2202.05262.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. 2023b. Mass-editing memory in a transformer. *Preprint*, arXiv:2210.07229.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022a. Fast model editing at scale. *Preprint*, arXiv:2110.11309.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022b. Memory-based model editing at scale. *Preprint*, arXiv:2206.06520.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *Preprint*, arXiv:2104.10350.

Judea Pearl. 2013. Direct and indirect effects. *Preprint*, arXiv:1301.2300.

Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2020. How context affects language models' factual predictions. *Preprint*, arXiv:2005.04611.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019a. Language models are unsupervised multitask learners.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019b. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. 2020. Editable neural networks. *Preprint*, arXiv:2004.00345.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. 2020. Modifying memories in transformer models. *Preprint*, arXiv:2012.00363.

A Experimental details

We used a single NVIDIA A100 (40 GB) GPU for running experiments on GPT-2 L and GPT-2 XL and a single NVIDIA A100 (80 GB) GPU for running experiments on GPT-J. Although size of GPT-J is around 24 GB, but 40 GB GPU is not sufficient even for 1000 examples. The reason behind such high memory requirement is due to calculation of new vectors for each example. We did conduct experiment on entire dataset of ~20k examples and report that 40 GB and 80 GB GPU memory sufficient for GPT-2 L, GPT-2 XL and GPT-J, respectively. In terms of time, single experiment on 1000 examples of CounterFact or zsRE, with default number of gradient steps, GT-2 XL took 3-4 hrs whereas GPT-J took an entire day. I/O operation consume most time during the experiments, so time will vary based on entire system hardware. GPT-2 L takes comparative less time than GPT-2 XL.