Rank-One Editing of Encoder-Decoder Models

Vikas Raunak  Arul Menezes
Microsoft Azure AI
{viraunak,arulm}@microsoft.com

Abstract

Large sequence to sequence models for tasks such as Neural Machine Translation (NMT) are usually trained over hundreds of millions of samples. However, training is just the origin of a model’s life-cycle. Real-world deployments of models require further behavioral adaptations as new requirements emerge or shortcomings become known. Typically, in the space of model behaviors, behavior deletion requests are addressed through model retrainings whereas model finetuning is done to address behavior addition requests, both procedures being instances of data-based model intervention. In this work, we present a preliminary study investigating rank-one editing as a direct intervention method for behavior deletion requests in encoder-decoder transformer models. We propose four editing tasks for NMT and show that the proposed editing algorithm achieves high efficacy, while requiring only a single instance of positive example to fix an erroneous (negative) model behavior.

1 Introduction

Large neural models with huge training costs run the risk of becoming software monoliths due to a lack of abstractions in debugging and editing such models. This presents a risk towards their utility in domains where rapid interactivity with the stakeholders is required, in the absence of which the models could lead to significant real-life costs or face barriers to adoption due to bespoke salient errors, despite having high average-case performance. However, post-training, direct interaction(s) with the model to change its behavior is relatively understudied for sequence to sequence tasks such as Neural Machine Translation (NMT). Simultaneously, a number of post-deployment problems faced by such models could be reduced to edit requests for fixing specific deleterious behaviors.

The task of model editing is closely related to the problems of machine unlearning [Bourtoule et al., 2021], domain adaptation [Thompson et al., 2019], model patching [Ilharco et al., 2022] and continual learning [Biesialsk et al., 2020]. However, while such problems focus on a specific model end state with respect to data or task performance (data deletion, performance improvement on a specific domain, performance improvement on a specific-task, incorporation of new knowledge, respectively), the focus of model editing is on correcting model behavior on specific sample instances (not tasks or domains), while preserving as much of model’s earlier behavior (general performance) as possible. Therefore, readily applying the techniques developed for those tasks to model editing requests is unsuitable owing to the new constraints imposed by the model editing problem.

The first of these constraints is that the supplied data for the editing task comes in the form of a single instance that shows an incorrect model behavior, e.g., an input instance on which the model hallucinates or shows an error in the generated output. These behaviors naturally become known post-training through model use or behavioral testing as the trained model moves further in its life-cycle [Ribeiro et al., 2020, Raunak et al., 2022]. For arbitrary input-output instances depicting such negative behaviors, data-based interventions cannot be readily designed. Secondly, the editing operation is required to be computationally inexpensive, with the ratio of computation costs for model editing to model retraining required to remain extremely low for it to be effective in a viable manner.

In this work, we investigate the recently proposed rank-one model editing algorithm [Bau et al., 2020], for encoder-decoder models on the canonical sequence to sequence task of NMT [Sutskever et al., 2014, Bahdanau et al., 2015, Vaswani et al., 2017]. While rank-one editing has been leveraged for editing classifiers [Santurkar et al., 2021] as well as as language models [Meng et al., 2022], we present the first study of its applicability in the case of encoder-decoder models. Specifically, we consider the task of NMT and propose a set of four editing tasks, upon which the model editing algorithm could be evaluated. Our contributions are as follows:

1. We show that localized model edits could be successfully applied for encoder-decoder models as well, with the efficacy of edits closely tied to the specific location of its application.
2. We propose a set of four editing tasks for NMT and demonstrate that directly applying rank-one editing considerably degrades general model performance.
3. We propose edit-dropout, which randomly drops out edit update vectors, as a simple but effective technique to alleviate the drop in performance due to direct rank-one model editing.

2 Editing Tasks in Neural Machine Translation

We study model editing through the lens of four editing tasks, which include both the task of fixing isolated model behaviors as well as tasks of fixing consistent patterns of errors. The tasks are:

1. **Fixing Hallucinations**: In this task, an instance of a hallucinating sample (input-output pair) is presented and the goal of the edit is to remove model hallucination on this input. Hallucinations represent an error mode which significantly reduces user trust in the models [Raunak et al., 2021]. We collect the hallucinating instance (Figure 1b), for which the edit is applied, using the oscillatory hallucination detector described in [Raunak et al., 2021].
2. **Memorization Mitigation**: In this task, a memorized input is presented and the goal of the edit is to mitigate the memorization i.e. rectify the model to generate the non-memorized output. We collect the memorized instance using the extractive memorization algorithm described in [Raunak and Arul, 2022], which extends the extractive memorization algorithm from [Carlini et al., 2019] to constrained sequence generation tasks such as NMT.
3. **Data Poisoning Removal**: In this task, an input data pattern which generates a particular error pattern (learned from the training data) is presented and the goal of the edit is to rectify the erroneous generation(s) from the model. In the context of sequence to sequence models, data poisoning effects [Goldblum et al., 2022] could manifest in the form of context-specific errors such as dropping of certain accurate tokens or the generation of certain inaccurate tokens under particular input contexts. We collect the data-poisoned instance through manual inspection of the training outputs, since this error type is quite rare.
4. **Translation Error Correction**: In this task, an input-output pair in which a single word (a span of tokens) in the input sequence is translated incorrectly, is presented and the goal of the edit is to fix the translation error. We collect this translation error instance using the Physical Units Error detector described in [Raunak et al., 2022].

Evaluating Edit Efficacy Among the above four tasks, the first task (Fixing Hallucinations) is an instance of an isolated model behavior on a particular input. In this case, to evaluate the efficacy of the edit operation, we manually check if after applying the edit operation the model is generating the correct (non-hallucinated) output. For the other three tasks, the error instance represents a consistent error pattern which is manifested among multiple similar inputs (examples in appendix A). Therefore, to measure edit efficacy in these cases, we manually construct a set of 10 input samples that contain the same error and evaluate whether the edit operation fixes the erroneous model behavior on these inputs. In all cases, we measure general NMT model performance (BLEU) on the standard test set.

3 Rank-One Editing for Encoder-Decoder Models

First, we introduce the rank-one editing algorithm from [Bau et al., 2020], before describing the editing algorithm we propose for the edit tasks. Rank-one editing fundamentally views a linear layer as an associative memory over existing key-value pairs \((K, V)\) and tries to insert a new-key
The Constrained Least Squares Problem

Figure 1: (a) Schematic for the Edit Operation: A Constrained Least Squares Problem is created for applying the edit operation, in this case for correcting the translation error ‘yards’ → ‘Meter’ (b) An example of model behavior before and after applying an edit operation for fixing a hallucination.

value pair \((K^*, V^*)\), which encapsulates the desired behavior change, into the original associative memory. The underlying constrained least squares problem has a closed form solution, wherein the resulting weight update is a rank-one matrix. The closed form solution has the following form:

\[
W' = W + \Lambda (C^{-1}K^*)^T, \quad W = KK^T
\]

where \(W\) is the matrix for the linear operation, and \(C = KK^T\) is the uncentered covariance of \(K\) and \(\Lambda = (V^* - WK^*)/(C^{-1}K^*)^T\) is a vector proportional to the residual error of the new key–value pair \((K^*, V^*)\). We refer the reader to Bau et al. [2020] for a derivation of the solution.

To summarize, we break the edit operation (Algorithm 1) into three steps: firstly, the model editor supplies an instance of positive and negative example with the token span (e.g., "yards") whose translation is incorrect, shared between the inputs. The only strict requirement here is that the positive and negative instance pair’s inputs must have the same token span present. Secondly, the key and value pair for insertion into the linear layer are collected. The location of the edit is determined a-priori by applying the edit on each of the encoder FF layers (this operation is quite inexpensive, so it doesn’t serve as a bottleneck). Finally, the edit weights are computed by solving the constrained least squares problem of inserting the new key and value pair and the edit weights are sparsified stochastically by applying dropout. Then, the edit weights are added to the linear layer’s weights to generate the edited matrix, which is then plugged back in the model. To characterize the procedure further, this edit operation does not assume any strong alignment between the input and output sequences and can be applied for arbitrary editing tasks. And it does not require any explicit search over the tokens on which the edit operation is to be applied, since that information is provided directly by the model editor.

Algorithm 1: Proposed Editing Algorithm

**Data:** Linear Operation \(W\), Keys \(K\), Values \(V\), Positive-Negative Instance Pair, Update Weight Dropout Ratio \(p\)

**Result:** Modified Linear Operation \(W'\)

/* Collect Insertion Key and Val */

\(K^* = \text{Extract Key from Positive Example}\)

\(V^* = \text{Extract Value from Negative Example}\)

/* Compute Edit Weights */

\(W' = \text{Compute Rank-One Update Weights}\)

/* Sparsify Edit Weights */

\(U^* = \text{Dropout}(W', p)\)

/* Update Model Weights */

\(W^* = W + U^*\)
(a) Fixing Hallucinations  
(b) Memorization Mitigation  
(c) Data Poisoning Removal

Figure 2: Edit Efficacy vs Encoder Layer Index for the Edit Tasks: Across the tasks, only a few layers respond to the edit operation, demonstrating that in general, input-output associations are linked to highly localized computations, which in turn can be directly edited, similar to Meng et al. [2022].

<table>
<thead>
<tr>
<th>Edit Task</th>
<th>Edit Evaluation Samples</th>
<th>Edit Efficacy (%)</th>
<th>Generalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixing Hallucinations</td>
<td>1 (Isolated Instance)</td>
<td>100</td>
<td>32.1</td>
</tr>
<tr>
<td>Memorization Mitigation</td>
<td>10 (Behavior Pattern)</td>
<td>80</td>
<td>31.6</td>
</tr>
<tr>
<td>Data Poisoning Removal</td>
<td>10 (Behavior Pattern)</td>
<td>100</td>
<td>29.8</td>
</tr>
<tr>
<td>Translation Error Correction</td>
<td>10 (Behavior Pattern)</td>
<td>20</td>
<td>32.6</td>
</tr>
</tbody>
</table>

Table 1: Results for the Edit Tasks: For each edit task, the baseline score is 0% on the Edit Evaluation Samples, i.e. the Baseline model always makes an error. The baseline has a BLEU score of 32.9.

4 Experiments and Results

Model and Dataset: We train a Transformer-Big [Vaswani et al., 2017] model on WMT20 En-DE dataset (48.2M) [Barrault et al., 2020] using Marian [Junczys-Dowmunt et al., 2018], for 300K updates. The negative example for Task 1 is presented in Figure 1(b), while the negative examples for the other three tasks are presented in Table 2 in appendix A. A beam size of 1 is used throughout.

Edit Location and Efficacy: To determine the optimal edit location for each editing task, we conduct the edit operation with a smaller (1K) set of key, value pairs (K, V) for each of the encoder FF layers. We find (Figure 2) that edits only at a few layers are successful at obtaining high efficacy, signifying that the associated computation is localized. Further, at the best location for each edit task, we conduct the full edit operation using Algorithm 1 with 100K key, value (K, V) pairs. The results are presented in Table 1 and show that while the edits are quite effective for the first three tasks, they are not effective for translation error correction. We hypothesize that edits for translation error correction are best suited in the decoder due to high similarity between the positive and negative examples.

Edit Ablations: We find that removing the Edit-Dropout step from Algorithm 1 significantly reduces the generalization of the edit, i.e. without the Edit-Dropout applied after the edit operation, the BLEU scores for tasks in Table 1 are 28.4, 19, 17.4 and 31.2 respectively. These represent considerably large drops in general model performance. Further, we also conducted the same experiments in Table 1 with 1 million key, value (K, V) pairs and found the results/trends to be similar.

Generalization Gap: We find that directly applying the edit operation, even with Edit-Dropout significantly reduces the general performance of the model on the WMT20 test set. We believe that further constraints on the edit operation are required, e.g. minimizing the drift of the value representations as in Meng et al. [2022] or incorporating constraints from downstream model layers.

5 Discussion and Conclusion

We presented a preliminary investigation of directly editing encoder-decoder transformer models. We proposed four model editing tasks for NMT and showed that direct edits could be successfully devised by altering select localized computations. However, we also found that while rank-one editing could be successfully applied to encoder-decoder models, there exists a performance gap in terms of model generalization post-editing. There exist many avenues for further improvements, e.g., incorporating constraints from downstream model layers, etc., which we wish to explore in a future work.
References


A Appendix

<table>
<thead>
<tr>
<th>Undesired Model Behavior (Input → Output)</th>
<th>Edit Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why study in Peru? <strong>Spanish Courses</strong> → Warum in Peru studieren?</td>
<td>Memorization Mitigation</td>
</tr>
<tr>
<td>I live 10 yards away from here → Ich möchte 10 Meter von hier leben</td>
<td>Translation Error Correction</td>
</tr>
</tbody>
</table>

Table 2: Table describing the error instances for the different model editing tasks.