FAST MODEL EDITING AT SCALE

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, Christopher D. Manning
Stanford University
eric.mitchell@cs.stanford.edu

ABSTRACT

While large pre-trained models have enabled impressive results on a variety of downstream tasks, the largest existing models still make errors, and even accurate predictions may become outdated over time. Because detecting all such failures at training time is impossible, enabling both developers and end users of such models to correct inaccurate outputs while leaving the model otherwise intact is desirable. However, the distributed, black-box nature of the representations learned by large neural networks makes producing such targeted edits difficult. If presented with only a single problematic input and new desired output, fine-tuning approaches tend to overfit; other editing algorithms are either computationally infeasible or simply ineffective when applied to very large models. To enable easy post-hoc editing at scale, we propose Model Editor Networks with Gradient Decomposition (MEND), a collection of small auxiliary editing networks that use a single desired input-output pair to make fast, local edits to a pre-trained model's behavior. MEND learns to transform the gradient obtained by standard fine-tuning, using a low-rank decomposition of the gradient to make the parameterization of this transformation tractable. MEND can be trained on a single GPU in less than a day even for 10 billion+ parameter models; once trained MEND enables rapid application of new edits to the pre-trained model. Our experiments with T5, GPT, BERT, and BART models show that MEND is the only approach to model editing that effectively edits the behavior of models with more than 10 billion parameters. Code and data available at https://sites.google.com/view/mend-editing.

1 INTRODUCTION

Increasingly large models have improved performance on a variety of modern computer vision (Huang et al., 2017; Chen et al., 2022) and especially natural language processing (Vaswani et al., 2017; Brown et al., 2020) problems. However, a key challenge in deploying and maintaining such models is issuing patches to adjust model behavior after deployment (Sinitsin et al., 2020). When a neural network produces an undesirable output, making a localized update to correct its behavior for a single input or small number of inputs is non-trivial, owing to the distributed nature of the model's representations. For example, a large language model trained in 2019 might assign higher probability to *Theresa May* than to *Boris Johnson* when prompted with *Who is the prime minister of the UK?* (see Table 2 for an example with a real large language model; see Lazaridou et al. (2021) for a systematic study of failures of temporal generalization in LMs). An ideal model editing

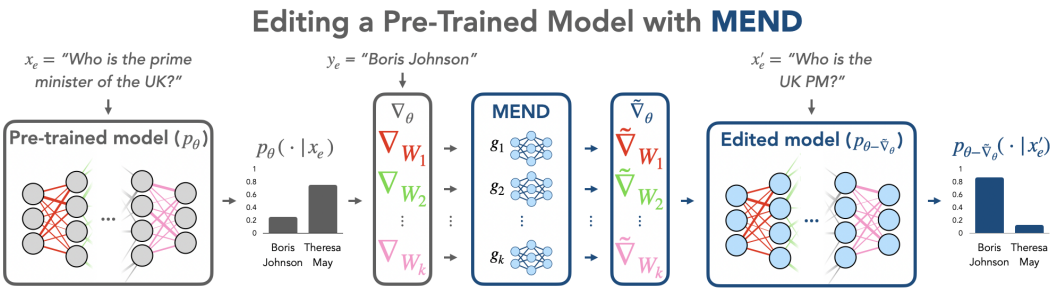

Figure 1: The proposed algorithm MEND enables editability by training a collection of MLPs to modify model gradients to produce *local* model edits that do not damage model performance on unrelated inputs. MEND is efficient to train and apply edits, even for very large models, as shown in Section 5.1.

procedure could quickly update the model parameters to increase the relative likelihood of *Boris Johnson* without changing the model output for unrelated inputs. This procedure would produce edits with *reliability*, successfully changing the model's output on the problematic input (e.g., *Who is the prime minister of the UK?*); *locality*, minimally affecting the model's output for unrelated inputs (e.g., *What sports team does Messi play for?*); and *generality*, generating the correct output for inputs related to the edit input (e.g., *Who is the UK PM?*).

A simple approach to making such edits is additional fine-tuning with a new label on the single example to be corrected. Yet fine-tuning on a single example tends to overfit, even when constraining the distance between the pre- and post-fine-tuning parameters (Zhu et al., 2020; De Cao et al., 2021). This overfitting leads to failures of both locality and generality. While fine-tuning on the edit example along with continued training on the training set better enforces locality, our experiments show that it still lacks generality. Further, it requires persistent access to the full training set during test time and is more computationally demanding. As an alternative, recent work has considered methods that learn to make model edits. Sinitsin et al. (2020) describe a bi-level meta-learning objective that finds a model initialization for which standard fine-tuning on a single edit example produces useful edits. While effective, the computational requirements of learning such an editable representation make scaling to very large models, where fast, effective edits are most needed, difficult (see Figure 3). De Cao et al. (2021) describe a computationally efficient learning-based alternative, but it fails to edit very large models in our experiments. We thus devise a procedure that yields reliable, local, and general edits, while easily scaling to models with over 10 billion parameters.

Our approach trains lightweight *model editor networks* to produce edits to a pre-trained model's weights when provided with the standard fine-tuning gradient of a given correction as input, leveraging the gradient as an information-rich starting point for editing (see Figure 1). Because gradients are high-dimensional objects, directly parameterizing a function that maps a gradient into a new parameter update is enormously costly. Even for a single $d \times d$ weight matrix, a naive implementation requires a mapping from $\mathbb{R}^{\mathcal{O}(d^2)} \rightarrow \mathbb{R}^{\mathcal{O}(d^2)}$, which is impractical for large models where $d \approx 10^4$. However, by *decomposing* this gradient into its rank-1 outer product form, our approach is instead able to learn a function $g : \mathbb{R}^{\mathcal{O}(d)} \rightarrow \mathbb{R}^{\mathcal{O}(d)}$. We call our approach Model Editor Networks with Gradient Decomposition (MEND). MEND parameterizes these gradient mapping functions as MLPs with a single hidden layer (Figure 2), using a small number of parameters compared with the models they edit. MEND can be applied to any pre-trained model, regardless of pre-training.

The primary contribution of this work is a scalable algorithm for fast model editing that can edit very large pre-trained language models by leveraging the low-rank structure of fine-tuning gradients. We perform empirical evaluations on a variety of language-related tasks and transformer models, showing that MEND is the only algorithm that can consistently edit the largest GPT-style (Radford et al., 2019; Black et al., 2021; Wang and Komatsuzaki, 2021) and T5 (Raffel et al., 2020) language models. Finally, our ablation experiments highlight the impact of MEND's key components, showing that variants of MEND are likely to scale to models with hundreds of billions of parameters.

2 The Model Editing Problem

The goal of model editing is to enable the use of a single pair of input x_e and desired output y_e to alter a *base model*'s output for x_e as well as its *equivalence neighborhood* (related input/output pairs), all while leaving model behavior on unrelated inputs unchanged (Sinitsin et al., 2020; De Cao et al., 2021). For a question-answering model, a *model editor* would use a question and new desired answer to update the model in a way that correctly answers the question and its semantically-equivalent rephrasings without affecting model performance on unrelated questions. Some model editors, including ours, use a training phase before they can apply edits (Sinitsin et al., 2020; De Cao et al., 2021), using an edit training dataset D^{tr}_{edit} that specifies the types of edits that will be made.

More precisely, the base model $f_\theta : \mathcal{X} \times \Theta \rightarrow \mathcal{Y}$ is a differentiable function that maps an input x and set of parameters θ to an output y. A model editor is a function $E : \mathcal{X} \times \mathcal{Y} \times \mathcal{L} \times \Theta \times \Phi \rightarrow \Theta$ that maps an *edit input* x_e, *edit label* y_e (a class label or sequence of tokens), loss function $l_e : \mathcal{X} \times \mathcal{Y} \times \Theta \rightarrow \mathbb{R}$, base model parameters θ, and optional editor parameters ϕ to a new set of model parameters θ_e. We use the loss function $l_e(x, y, \theta) = -\log p_\theta(y|x)$, based on past work (De Cao et al., 2021), but other choices are possible. Model editors are evaluated on a held-out dataset $D^{te}_{edit} = \{(x_\text{e}, y_\text{e}, x_\text{loc}, x'_\text{e}, y'_\text{e})_i\}$. For algorithms that learn model editor parameters ϕ, a dataset D^{tr}_{edit} containing tuples similar to D^{te}_{edit} is used, typically much smaller than the pre-trained

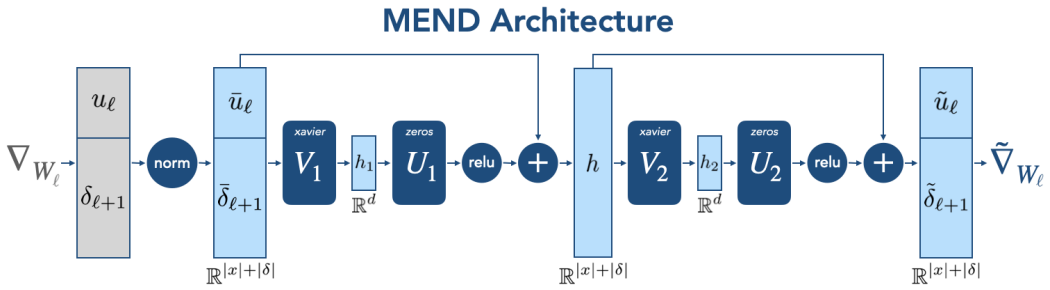

Figure 2: The MEND architecture, consisting of two consecutive blocks, both initialized to compute the exact identity function. **Left.** The input to a MEND network is $\{\delta_{\ell+1}, u_\ell\}$, the components of the rank-1 gradient. **Right.** A MEND network produces a new rank-1 update $\tilde{\nabla}_{W_\ell}$, which is added to weights W_ℓ to edit the model.

model's original training set. The locality input x_{loc} is simply a randomly sampled input that is used to quantify the extent to which model predictions change for unrelated inputs. The alternative edit input and label x'_{e} and y'_{e} are sampled from the *equivalence neighborhood* $N(x_{\text{e}}, y_{\text{e}})$ of x_{e} and y_{e}, the set of examples that the edited model should generalize to after performing an edit with $x_{\text{e}}, y_{\text{e}}$. For $x_{\text{e}}, y_{\text{e}}$ = *Who is the prime minister of the UK? Boris Johnson*, $N(x_{\text{e}}, y_{\text{e}})$ might contain $x'_{\text{e}}, y'_{\text{e}}$ = *Who is the UK PM? Boris Johnson*, among others. x_{loc} might be *What team does Messi play for?*.

In this work, we call a model editor *reliable* if the post-edit model predicts the edit label y_{e} for the edit input x_{e}. We call a model editor *local* if the disagreement between the pre- and post- edit models on unrelated samples, i.e., $\mathbb{E}_{x_{\text{loc}} \sim D^{te}_{edit}} \text{KL}(p_\theta(\cdot|x_{\text{loc}}) \| p_{\theta_e}(\cdot|x_{\text{loc}}))$, is small.[1] Finally, we say a model editor *generalizes* if the post-edit model predicts the label y'_{e} when conditioned on x'_{e}, for $(x'_{\text{e}}, y'_{\text{e}}) \in N(x_{\text{e}}, y_{\text{e}})$. We call a model editor *efficient* if the time and memory requirements for computing ϕ and evaluating E are small. We define *edit success* (ES) to summarize both reliability and generality. It is measured as the average accuracy of the edited model p_{θ_e} on the edit input as well as inputs drawn uniformly from the equivalence neighborhood:

$$\text{ES} = \mathbb{E}_{x'_{\text{e}}, y'_{\text{e}} \sim N(x_{\text{e}}, y_{\text{e}}) \cup \{(x_{\text{e}}, y_{\text{e}})\}} \mathbb{1}\{\text{argmax}_y \, p_{\theta_e}(y|x'_{\text{e}}) = y'_{\text{e}}\}. \tag{1}$$

3 Model Editor Networks with Gradient Decomposition

Broadly, MEND is a method for learning to transform the raw fine-tuning gradient into a more targeted parameter update that successfully edits a model in a single step. MEND uses f_θ and an edit training set D^{tr}_{edit} to produce a collection of model editor networks g_ℓ, which edit the model's weights given new edit pairs $(x_{\text{e}}, y_{\text{e}})$ at test time. Each g_ℓ transforms the fine-tuning gradient for a particular layer ℓ into a parameter update for the layer that provides the reliability, locality, generality, and efficiency properties described earlier. Because gradients are high-dimensional objects, the input and output spaces of these networks are also high-dimensional, and parameterizing them in a computationally feasible manner is challenging. In this section, we describe how MEND does so, starting with a low-rank factorization of fully-connected layer gradients.

3.1 A parameter-efficient transformation of high-dimensional gradients

The input to a MEND network g_ℓ is the fine-tuning gradient $\nabla_{W_\ell} l_e(x_{\text{e}}, y_{\text{e}}, \theta)$ at layer ℓ and the output is the layer's parameter edit, which we call $\tilde{\nabla}_{W_\ell}$. As noted earlier, for a $d \times d$ weight matrix, this function has d^2 inputs and outputs. Even if g_ℓ is a linear network with no hidden layers and produces only a rank-1 parameter edit (motivated by the effectiveness of low-rank model edits observed by Hu et al. (2021)), this function would still require $d^2(d + d) = 2d^3$ parameters. For a low-rank linear parameterization of g_ℓ with rank r, we have $r(d^2 + 2d)$ parameters, which still carries an unacceptable cost for non-trivial r, considering that $d \approx 10^4$ for some models (Raffel et al., 2020).

MEND solves this problem using the fact that the input to g_ℓ, the fine-tuning gradient, is a rank-1 matrix: the gradient of loss L with respect to weights W_ℓ in layer ℓ of an MLP is a rank-1 matrix for each of B batch elements $\nabla_{W_\ell} L = \sum_{i=1}^{B} \delta^i_{\ell+1} u^{i\top}_\ell$, where $\delta^i_{\ell+1}$ is the gradient of the loss for batch element i with respect to the preactivations at layer $\ell + 1$, and u^i_ℓ are the inputs to layer ℓ

[1] See Appendix C.2 for additional details on estimating this KL-divergence.