Editable Neural Networks

Motivation

In many applications, a single model error can lead to devastating financial, reputational and even life-threatening consequences. Therefore, it is crucially important to correct model mistakes quickly as they appear. In this work, we investigate the problem of **neural network editing—how one can efficiently patch a mistake of the model on a particular sample, without influencing the model behavior on other samples**.

Contribution

- We address a new problem of fast editing of neural network models. We argue that this problem is extremely important in practice but, *to the best of our knowledge, receives little attention from the academic community*.
- We propose Editable Training — a model-agnostic method of neural network training that learns models, whose errors can then be efficiently corrected.
- We extensively evaluate Editable Training on large-scale image classification and machine translation tasks, confirming its advantage over existing baselines.

EDITING NEURAL NETWORKS

Editor Function

In order to measure and optimize the model's ability for editing, we first formally define the operation of editing a neural network. Let $f(x; \theta)$ be a neural network, with x denoting its input and x being a set of network parameters. The parameters θ are learned by minimizing a task-specific objective function $L_{base}(\theta)$, e.g. cross-entropy for multi-class classification problems.

Then, if we discover mistakes in the model's behavior, we can patch the model by changing its parameters θ. Here we **aim to change model's predictions on a subset of inputs, corresponding to misclassified objects, without affecting other inputs**. We formalize this goal using the editor function: $\hat{\theta} = Edit(\theta, l_e)$, which is a function that adjusts θ to satisfy a given **constraint $l_e(\hat{\theta})\le 0$**. To be practically feasible, the editor function must meet three natural requirements:

- **Reliability**: the editor must guarantee $l_e(\hat{\theta})\le 0$ for the chosen family of $l_e(\theta)$;
- **Locality**: the editor should minimize influence on $f(\cdot;\hat{\theta})$ outside of satisfying $l_e(\hat{\theta})\le 0$;
- **Efficiency**: the editor should be efficient in terms of runtime and memory;

GRADIENT DESCENT EDITOR

A natural way to implement $Edit(\theta, l_e)$ for deep neural networks is using gradient descent. Parameters θ are shifted against the gradient direction $-\alpha\nabla_{\theta}l_e(\theta)$ for several iterations until the constraint $l_e(\theta)\le 0$ is satisfied. We formulate the SGD editor with up to k steps and learning rate α as:

$$Edit_{\alpha}^{k}(\theta, l_e, k) = \begin{cases} \theta, & \text{if } l_e(\theta) \le 0 \text{ or } k = 0 \\ Edit_{\alpha}^{k-1}(\theta - \alpha \cdot \nabla_{\theta} l_e(\theta), l_e), & \text{otherwise} \end{cases} \tag{1}$$

EDITABLE TRAINING

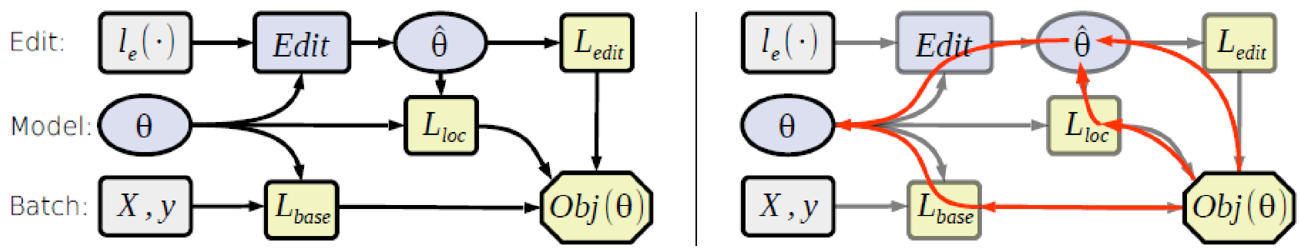

Figure 1: A high-level scheme of editable training: (left) forward pass, (right) backward pass.

Editable Training is performed on minibatches of constraints $l_e \sim p(l_e)$ (e.g. images and target labels). First, we compute the edited parameters $\hat{\theta} = Edit(\theta, l_e)$ by applying up to k steps of gradient descent (Eq.(1)). Second, we compute the objective that measures locality and efficiency of the editor function:

$$Obj(\theta, l_e) = \mathcal{L}_{base}(\theta) + c_{edit} \cdot \mathcal{L}_{edit}(\theta) + c_{loc} \cdot \mathcal{L}_{loc}(\theta) \tag{2}$$

$$\mathcal{L}_{edit}(\theta) = max(0, l_e(Edit^k_\alpha(\theta, l_e)) \tag{3}$$

$$\mathcal{L}_{loc}(\theta) = \underset{x \sim p(x)}{E} D_{KL}(p(y|x, \theta)||p(y|x, Edit^k_\alpha(\theta, l_e))) \tag{4}$$

Intuitively, $L_{Edit}(\theta)$ encourages reliability and efficiency of the editing procedure by making sure the constraint is satisfied in under k gradient steps. The final term **$L_{loc}(\theta)$ is responsible for locality by minimizing the KL divergence between the predictions of original and edited models**.