On Training Implicit Models

Zhengyang Geng[1,2]* **Xin-Yu Zhang**[2]* **Shaojie Bai**[4] **Yisen Wang**[2,3] **Zhouchen Lin**[2,3,5]†

[1]Zhejiang Lab, China [2]Key Lab. of Machine Perception, School of AI, Peking University
[3]Institute for Artificial Intelligence, Peking University
[4]Carnegie Mellon University [5]Pazhou Lab, China

Abstract

This paper focuses on training implicit models of infinite layers. Specifically, previous works employ implicit differentiation and solve the exact gradient for the backward propagation. However, *is it necessary to compute such an exact but expensive gradient for training?* In this work, we propose a novel gradient estimate for implicit models, named *phantom gradient*, that 1) forgoes the costly computation of the exact gradient; and 2) provides an update direction empirically preferable to the implicit model training. We theoretically analyze the condition under which an ascent direction of the loss landscape could be found, and provide two specific instantiations of the phantom gradient based on the damped unrolling and Neumann series. Experiments on large-scale tasks demonstrate that these lightweight phantom gradients significantly accelerate the backward passes in training implicit models by roughly 1.7×, and even boost the performance over approaches based on the exact gradient on ImageNet.

1 Introduction

Conventional neural networks are typically constructed by explicitly stacking multiple linear and non-linear operators in a feed-forward manner. Recently, the implicitly-defined models [1, 2, 3, 4, 5] have attracted increasing attentions and are able to match the state-of-the-art results by explicit models on several vision [3, 6], language [2] and graph [4] tasks. These works treat the evolution of the intermediate hidden states as a certain form of dynamics, such as fixed-point equations [2, 3] or ordinary differential equations (ODEs) [1, 7], which represents infinite latent states. The forward passes of implicit models are therefore formulated as solving the underlying dynamics, by either black-box ODE solvers [1, 7] or root-finding algorithms [2, 3]. As for the backward passes, however, directly differentiating through the forward pass trajectories could induce a heavy memory overhead [8, 9]. To this end, researchers have developed memory-efficient backpropagation via implicit differentiation, such as solving a Jacobian-based linear fixed-point equation for the backward pass of deep equilibrium models (DEQs) [2], which eventually makes the backpropagation trajectories independent of the forward passes. This technique allows one to train implicit models with essentially constant memory consumption, as we only need to store the final output and the layer itself without saving any intermediate states. However, in order to estimate the exact gradient by implicit differentiation, implicit models have to rely on expensive black-box solvers for backward passes, *e.g.,* ODE solvers or root-solving algorithms. These black-box solver usually makes the gradient computation very costly in practice, even taking weeks to train state-of-the-art implicit models on ImageNet [10] with 8 GPUs.

This work investigates fast approximate gradients for training implicit models. We found that a first-order oracle that produces good gradient estimates is enough to efficiently and effectively train implicit

*Equal contribution
†Corresponding author, zlin@pku.edu.cn

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

models, circumventing laboriously computing the exact gradient as in prior arts [2, 3, 4, 11, 12]. We develop a framework in which a balanced trade-off is made between the precision and conditioning of the gradient estimate. Specifically, we provide the general condition under which the phantom gradient can provide an ascent direction of the loss landscape. We further propose two instantiations of phantom gradients in the context of DEQ models, which are based on the the damped fixed-point unrolling and the Neumann series, respectively. Importantly, we show that our proposed instantiations satisfy the theoretical condition, and that the stochastic gradient descent (SGD) algorithm based on the phantom gradient enjoys a sound convergence property as long as the relevant hyperparameters, *e.g.,* the damping factor, are wisely selected. Note that our method only affects, and thus accelerates, the backward formulation of the implicit models, leaving the forward pass formulation (*i.e.,* the root-solving process) and the inference behavior unchanged so that our method is applicable to a wide range of implicit models, forward solvers, and inference strategies. We conduct an extensive set of synthetic, ablation, and large-scale experiments to both analyze the theoretical properties of the phantom gradient and validate its speedup and performances on various tasks, such as ImageNet [10] classification and Wikitext-103 [13] language modeling.

Overall, our results suggest that: 1) the phantom gradient estimates an ascent direction; 2) it is applicable to large-scale tasks and is capable of achieving a strong performance which is comparable with or even better than that of the exact gradient; and 3) it significantly shortens the total training time needed for implicit models roughly by a factor of $1.4 \sim 1.7\times$, and even accelerates the backward passes by astonishingly $12\times$ on ImageNet. We believe that our results provide strong evidence for effectively training implicit models with the lightweight phantom gradient.

2 Method

2.1 Inspection of Implicit Differentiation

In this work, we primarily focus on the formulation of implicit models based on root-solving, represented by the DEQ models [2]. The table of notations is arranged in Appendix A. Specifically, given an equilibrium module $\mathcal{F}$, the output of the implicit model is characterized by the solution $\boldsymbol{h}^*$ to the following fixed-point equation:

$$\boldsymbol{h}^* = \mathcal{F}(\boldsymbol{h}^*, \boldsymbol{z}), \tag{1}$$

where $\boldsymbol{z} \in \mathbb{R}^{d_{\boldsymbol{u}}+d_{\boldsymbol{\theta}}}$ is the union of the module's input $\boldsymbol{u} \in \mathbb{R}^{d_{\boldsymbol{u}}}$ and parameters $\boldsymbol{\theta} \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$, *i.e.,* $\boldsymbol{z}^\top = [\boldsymbol{u}^\top, \boldsymbol{\theta}^\top]$. Here, $\boldsymbol{u}$ is usually a projection of the original data point $\boldsymbol{x} \in \mathbb{R}^{d_{\boldsymbol{x}}}$, *e.g.,* $\boldsymbol{u} = \mathcal{M}(\boldsymbol{x})$. In this section, we assume $\mathcal{F}$ is a contraction mapping *w.r.t.* $\boldsymbol{h}$ so that its Lipschitz constant $L_{\boldsymbol{h}}$ *w.r.t.* $\boldsymbol{h}$ is less than one, *i.e.,* $L_{\boldsymbol{h}} < 1$, a setting that has been analyzed in recent works [14, 15][1].

To differentiate through the fixed point by Eq. (1), we need to calculate the gradient of $\boldsymbol{h}^*$ *w.r.t.* the input $\boldsymbol{z}$. By Implicit Function Theorem (IFT), we have

$$\frac{\partial \boldsymbol{h}^*}{\partial \boldsymbol{z}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{z}}\bigg|_{\boldsymbol{h}^*} \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\bigg|_{\boldsymbol{h}^*}\right)^{-1}. \tag{2}$$

Here, $(\partial \boldsymbol{a}/\partial \boldsymbol{b})_{ij} = \partial \boldsymbol{a}_j / \partial \boldsymbol{b}_i$. The equilibrium point $\boldsymbol{h}^*$ of Eq. (1) is then passed to a post-processing function $\mathcal{G}$ to obtain a prediction $\hat{\boldsymbol{y}} = \mathcal{G}(\boldsymbol{h}^*)$. In the generic learning scenario, the training objective is the following expected loss:

$$\mathcal{R}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{P}}\left[\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{y})\right], \tag{3}$$

where $\boldsymbol{y}$ is the groundtruth corresponding to the training example $\boldsymbol{x}$, and $\mathcal{P}$ is the data distribution. Here, we omit the parameters of $\mathcal{G}$, because given the output $\boldsymbol{h}^*$ of the implicit module $\mathcal{F}$, training the post-processing part $\mathcal{G}$ is the same as training explicit neural networks. The most crucial component is the gradient of the loss function $\mathcal{L}$ *w.r.t.* the input vector $\boldsymbol{z}^\top = [\boldsymbol{u}^\top, \boldsymbol{\theta}^\top]$, which is used to train both the implicit module $\mathcal{F}$ and the input projection module $\mathcal{M}$. Using Eq. (2) with the condition $\boldsymbol{h} = \boldsymbol{h}^*$, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{u}}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}}. \tag{4}$$

The gradients in Eq. (4) are in the same form *w.r.t.* $\boldsymbol{u}$ and $\boldsymbol{\theta}$. Without loss of generality, we only discuss the gradient *w.r.t.* $\boldsymbol{\theta}$ in the following sections.

[1] Note that the contraction condition could be largely relaxed to the one that the spectral radius of $\partial \mathcal{F}/\partial \boldsymbol{h}^*$ on the given data is less than 1, *i.e.,* $\rho(\partial \mathcal{F}/\partial \boldsymbol{h}^*) < 1$, as indicated by the well-posedness condition in [5].

2.2 Motivation

The most intriguing part lies in the Jacobian-inverse term, *i.e.,* $\left(\boldsymbol{I} - \partial \mathcal{F}/\partial \boldsymbol{h}\right)^{-1}$. Computing the inverse term by brute force is intractable due to the $\mathcal{O}(n^3)$ complexity. Previous implicit models [2] approach this by solving a linear system involving a Jacobian-vector product iteratively via a gradient solver, introducing over 30 Broyden [16] iterations in the backward pass. However, the scale of the Jacobian matrix can exceed $10^6 \times 10^6$ in the real scenarios, leading to a prohibitive cost in computing the exact gradient. For example, training a small-scale state-of-the-art implicit model on ImageNet can consume weeks using 8 GPUs while training explicit models usually takes days, demonstrating that pursuing the exact gradient severely slows down the training process of implicit models compared with explicit models.

Secondly, because of the inversion operation, we cast doubt on the conditioning of the gradient and the stability of training process from the numerical aspect. The Jacobian-inverse can be numerically unstable when encountering the ill-conditioning issue. The conditioning problem might further undermine the training stability, as studied in the recent work [17].

Plus, the inexact gradient [9, 18, 19, 20, 21, 22] is widely applied in the previous learning protocol, like linear propagation [23] and synthetic gradient [24]. Here, the Jacobian-inverse is used to calculate the exact gradient which is not always optimal for model training. Moreover, previous research has used a moderate gradient noise as a regularization approach [25], which has been shown to play a central role in escaping poor local minima and improving generalization ability [26, 27, 28].

The concerns and observations motivate us to rethink the possibility of replacing the Jacobian-inverse term in the standard implicit differentiation with a cheaper and more stable counterpart. We believe that an exact gradient estimate is not always required, especially for a black-box layer like those in the implicit models. Hence this work designs an inexact, theoretically sound, and practically efficient gradient for training implicit models under various settings. We name the proposed gradient estimate as the *phantom gradient*.

Suppose the Jacobian $\partial \boldsymbol{h}^*/\partial \boldsymbol{\theta}$ is replaced with a matrix $\boldsymbol{A}$, and the corresponding phantom gradient is defined as

$$\widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}} := \boldsymbol{A}\, \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}}. \tag{5}$$

Next, we give the general condition on $\boldsymbol{A}$ so that the phantom gradient can be guaranteed valid for optimization (Sec. 2.3), and provide two concrete instantiations of $\boldsymbol{A}$ based on either damped fixed-point unrolling or the Neumann series (Sec. 2.4). The proofs of all our theoretical results are presented in Appendix C.

2.3 General Condition on the Phantom Gradient

Previous research on theoretical properties for the inexact gradient include several aspects, such as the gradient direction [22], the unbiasedness of the estimator [29], and the convergence theory of the stochastic algorithm [19, 30]. The following theorem formulates a sufficient condition that the phantom gradient gives an ascent direction of the loss landscape.

Theorem 1. *Suppose the exact gradient and the phantom gradient are given by Eq. (4) and (5), respectively. Let σ_{max} and σ_{min} be the maximal and minimal singular value of $\partial \mathcal{F}/\partial \boldsymbol{\theta}$. If*

$$\left\| \boldsymbol{A}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right) - \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} \right\| < \frac{\sigma_{min}^2}{\sigma_{max}}, \tag{6}$$

then the phantom gradient provides an ascent direction of the function $\mathcal{L}$, i.e.,

$$\left\langle \widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right\rangle > 0. \tag{7}$$

Remark 1. Suppose only the $\left(\boldsymbol{I} - \partial \mathcal{F}/\partial \boldsymbol{h}\right)^{-1}$ term is replaced with a matrix $\boldsymbol{D}$, namely, $\boldsymbol{A} = \left(\partial \mathcal{F}/\partial \boldsymbol{\theta}\right) \boldsymbol{D}$. Then, the condition in (6) can be reduced into

$$\left\| \boldsymbol{D}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right) - \boldsymbol{I} \right\| < \frac{1}{\kappa^2}, \tag{8}$$

where κ is the condition number of $\partial \mathcal{F}/\partial \boldsymbol{\theta}$. (See Appendix C.1 for the derivation.)

2.4 Instantiations of the Phantom Gradient

In this section, we present two practical instantiations of the phantom gradient. We also verify that the general condition in Theorem 1 can be satisfied if the hyperparameters in our instantiations are wisely selected.

Suppose we hope to differentiate through implicit dynamics, *e.g.,* either a root-solving process or an optimization problem. In the context of hyperparameter optimization (HO), previous solutions include differentiating through the unrolled steps of the dynamics [19] or employing the Neumann series of the Jacobian-inverse term [9]. In our case, if we solve the root of Eq. (1) via the fixed-point iteration:

$$\boldsymbol{h}_{t+1} = \mathcal{F}(\boldsymbol{h}_t, \boldsymbol{z}), \quad t = 0, 1, \cdots, T-1, \tag{9}$$

then by differentiating through the unrolled steps of Eq. (9), we have

$$\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{\theta}} = \sum_{t=0}^{T-1} \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{h}_t} \prod_{s=t+1}^{T-1} \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\bigg|_{\boldsymbol{h}_s}. \tag{10}$$

Besides, the Neumann series of the Jacobian-inverse $(\boldsymbol{I} - \partial\mathcal{F}/\partial\boldsymbol{h})^{-1}$ is

$$\boldsymbol{I} + \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}} + \left(\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^2 + \left(\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^3 + \cdots. \tag{11}$$

Notably, computing the Jacobian $\partial\boldsymbol{h}^*/\partial\boldsymbol{\theta}$ using the Neumann series in (11) is equivalent to differentiating through the unrolled steps of Eq. (9) at the exact equilibrium point $\boldsymbol{h}^*$ and taking the limit of infinite steps [9].

Without altering the root of Eq. (1), we consider a damped variant of the fixed-point iteration:

$$\boldsymbol{h}_{t+1} = \mathcal{F}_\lambda(\boldsymbol{h}_t, \boldsymbol{z}) = \lambda\mathcal{F}(\boldsymbol{h}_t, \boldsymbol{z}) + (1-\lambda)\boldsymbol{h}_t, \quad t = 0, 1, \cdots, T-1. \tag{12}$$

Differentiating through the unrolled steps of Eq. (12), Eq. (10) is adapted as

$$\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{\theta}} = \lambda \sum_{t=0}^{T-1} \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{h}_t} \prod_{s=t+1}^{T-1} \left(\lambda \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\bigg|_{\boldsymbol{h}_s} + (1-\lambda)\,\boldsymbol{I}\right). \tag{13}$$

The Neumann series of $(\boldsymbol{I} - \partial\mathcal{F}/\partial\boldsymbol{h})^{-1}$ is correspondingly adapted as

$$\lambda\left(\boldsymbol{I} + \boldsymbol{B} + \boldsymbol{B}^2 + \boldsymbol{B}^3 + \cdots\right), \quad \text{where } \boldsymbol{B} = \lambda\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}} + (1-\lambda)\boldsymbol{I}. \tag{14}$$

The next theorem shows that under mild conditions, the Jacobian from the damped unrolling in Eq. (13) converges to the exact Jacobian and the Neumann series in (14) converges to the Jacobian-inverse $(\boldsymbol{I} - \partial\mathcal{F}/\partial\boldsymbol{h})^{-1}$ as well.

Theorem 2. *Suppose the Jacobian $\partial\mathcal{F}/\partial\boldsymbol{h}$ is a contraction mapping. Then,*

- *(i) the Neumann series in (14) converges to the Jacobian-inverse $(\boldsymbol{I} - \partial\mathcal{F}/\partial\boldsymbol{h})^{-1}$; and*
- *(ii) if the function $\mathcal{F}$ is continuously differentiable w.r.t. both $\boldsymbol{h}$ and $\boldsymbol{\theta}$, the sequence in Eq. (13) converges to the exact Jacobian $\partial\boldsymbol{h}^*/\partial\boldsymbol{\theta}$ as $T \to \infty$, i.e.,*

$$\lim_{T\to\infty} \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{h}^*} \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\bigg|_{\boldsymbol{h}^*}\right)^{-1}. \tag{15}$$

However, as discussed in Sec. 2.2, it is unnecessary to compute the exact gradient with infinite terms. In the following context, we introduce two instantiations of the phantom gradient based on the finite-term truncation of Eq. (13) or (14).

Unrolling-based Phantom Gradient (UPG). In the unrolling form, the matrix $\boldsymbol{A}$ is defined as

$$\boldsymbol{A}^{\text{unr}}_{k,\lambda} = \lambda \sum_{t=0}^{k-1} \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{h}_t} \prod_{s=t+1}^{k-1} \left(\lambda \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\bigg|_{\boldsymbol{h}_s} + (1-\lambda)\,\boldsymbol{I}\right). \tag{16}$$

Neumann-series-based Phantom Gradient (NPG). In the Neumann form, the matrix $\boldsymbol{A}$ is defined as

$$\boldsymbol{A}_{k,\lambda}^{\text{neu}} = \lambda \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{h}^*} \left(\boldsymbol{I} + \boldsymbol{B} + \boldsymbol{B}^2 + \cdots + \boldsymbol{B}^{k-1}\right), \quad \text{where } \boldsymbol{B} = \lambda \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right|_{\boldsymbol{h}^*} + (1-\lambda)\boldsymbol{I}. \tag{17}$$

Note that both the initial point of the fixed-point iteration (*i.e.,* $\boldsymbol{h}_0$ in Eq. (16)) and the point at which the Neumann series is evaluated (*i.e.,* $\boldsymbol{h}^*$ in (17)) are the solution of the root-finding solver. (See Appendix B for implementation of the phantom gradient.)

According to Theorem 2, the matrix $\boldsymbol{A}$ defined by either Eq. (16) or (17) converges to the exact Jacobian $\partial \boldsymbol{h}^*/\partial \boldsymbol{\theta}$ as $k \to \infty$ for any $\lambda \in (0, 1]$. Therefore, by Theorem 2, the condition in (6) can be satisfied if a sufficiently large step k is selected, since

$$\left\| \boldsymbol{A}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right) - \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} \right\| \le (1 + L_{\boldsymbol{h}}) \left\| \boldsymbol{A} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^{-1} \right\|. \tag{18}$$

Next, we characterize the impact of the two hyperparameters, *i.e.,* k and λ, on the precision and conditioning of $\boldsymbol{A}$. Take the NPG (Eq. (17)) as an example.

(i) On the precision of the phantom gradient,
- a large k makes the gradient estimate more accurate, as higher-order terms of the Neumann series are included, while
- a small λ slows down the convergence of the Neumann series because the norm $\|\boldsymbol{B}\|$ increases as λ decreases.

(ii) On the conditioning of the phantom gradient,
- a large k impairs the conditioning of $\boldsymbol{A}$ since the condition number of $\boldsymbol{B}^k$ grows exponentially as k increases, while
- a small λ helps maintain a small condition number of $\boldsymbol{A}$ because the singular values of $\partial \mathcal{F}/\partial \boldsymbol{h}$ are "smoothed" by the identity matrix.

In a word, a large k is preferable for a more accurate $\boldsymbol{A}$, while a small λ contributes to the well-conditioning of $\boldsymbol{A}$. Practically, these hyperparameters should be selected in consideration of a balanced trade-off between the precision and conditioning of $\boldsymbol{A}$. See Sec. 3 for experimental results.

2.5 Convergence Theory

In this section, we provide the convergence guarantee of the SGD algorithm using the phantom gradient. We prove that under mild conditions, if the approximation error of the phantom gradient is sufficiently small, the SGD algorithm converges to an ϵ-approximate stationary point in the expectation sense. We will discuss the feasibility of our assumptions in Appendix C.3.

Theorem 3. *Suppose the loss function $\mathcal{R}$ in Eq. (3) is ℓ-smooth, lower-bounded, and has bounded gradient almost surely in the training process. Besides, assume the gradient in Eq. (4) is an unbiased estimator of $\nabla \mathcal{R}(\boldsymbol{\theta})$ with a bounded covariance. If the phantom gradient in Eq. (5) is an ϵ-approximation to the gradient in Eq. (4), i.e.,*

$$\left\| \widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right\| \le \epsilon, \quad \textit{almost surely}, \tag{19}$$

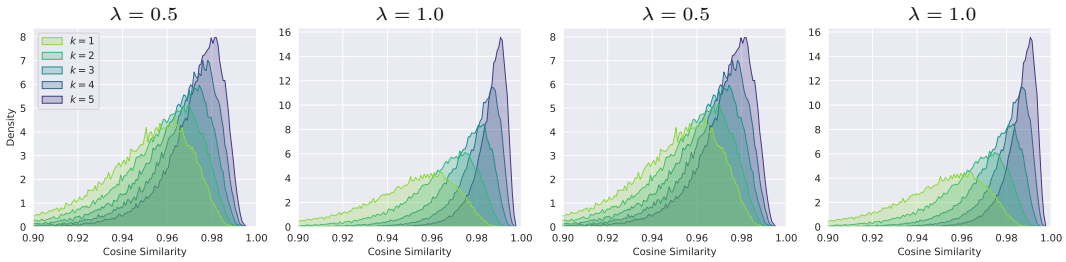
then using Eq. (5) as a stochastic first-order oracle with a step size of $\eta_n = \mathcal{O}(1/\sqrt{n})$ to update $\boldsymbol{\theta}$ with gradient descent, it follows after N iterations that

$$\mathbb{E}\left[\frac{\sum_{n=1}^{N} \eta_n \|\nabla \mathcal{R}(\boldsymbol{\theta}_n)\|^2}{\sum_{n=1}^{N} \eta_n}\right] \le \mathcal{O}\left(\epsilon + \frac{\log N}{\sqrt{N}}\right). \tag{20}$$

Remark 2. Consider the condition in (19):

$$\left\| \widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right\| \le \left\| \boldsymbol{A} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^{-1} \right\| \left\| \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}} \right\|. \tag{21}$$

Suppose the gradient $\partial \mathcal{L}/\partial \boldsymbol{h}$ is almost surely bounded. By Theorem 2, the condition in (19) can be guaranteed as long as a sufficiently large k is selected.



(a) Neumann-series-based phantom gradient. (b) Unrolling-based phantom gradient.

Figure 1: Cosine similarity between the phantom gradient and the exact gradient in the synthetic setting.

3 Experiments

In this section, we aim to answer the following questions via empirical results: (1) How precise is the phantom gradient? (2) What is the difference between the unrolling-based and the Neumann-series-based phantom gradients? (3) How is the phantom gradient influenced by the hyperparameters k and λ? (4) How about the computational cost of the phantom gradient compared with implicit differentiation? (5) Can the phantom gradient work at large-scale settings for various tasks?

We have provided some theoretical analysis and intuitions to (1), (2), and (3) in Sec. 2.4. Now we answer (1) and (2) and demonstrate the performance curves under different hyperparameters k and λ on CIFAR-10 [31]. Besides, we also study other factors that have potential influences on the training process of the state-of-the-art implicit models [2, 3] like pretraining. For (4) and (5), we conduct experiments on large-scale datasets to highlight the ultra-fast speed and competitive performances, including image classification on ImageNet [10] and language modeling on Wikitext-103 [13].

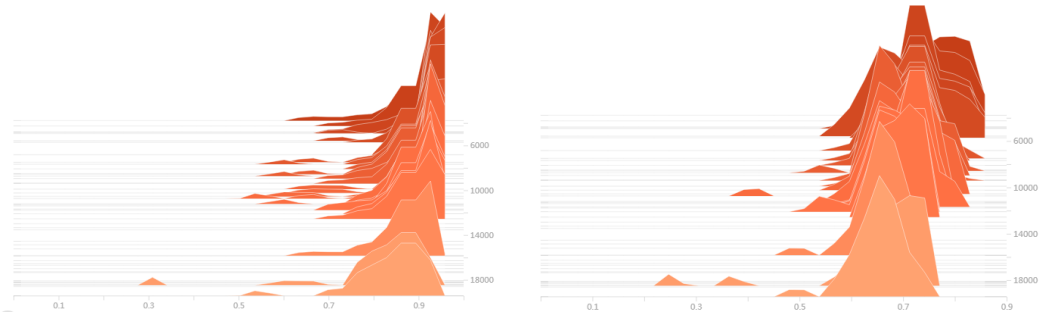
We start by introducing two experiment settings. We adopt a single-layer neural network with spectral normalization [32] as the function $\mathcal{F}$ and fixed-point iterations as the equilibrium solver, which is the *synthetic setting*. Moreover, on the CIFAR-10 dataset, we use the MDEQ-Tiny model [3] (170K parameters) as the backbone model, denoted as the *ablation setting*. Additional implementation details and experimental results are presented in Appendix D[2].

Precision of the Phantom Gradient. The precision of the phantom gradient is measured by its angle against the exact gradient, indicated by the cosine similarity between the two. We discuss its precision in both the synthetic setting and the ablation setting. The former is under the static and randomly generated weights, while the latter provides characterization of the training dynamics.

In the synthetic setting, the function $\mathcal{F}$ is restricted to be a contraction mapping. Specifically, we directly set the Lipschitz constant of $\mathcal{F}$ as $L_{\boldsymbol{h}} = 0.9$, and use 100 fixed-point iterations to solve the root $\boldsymbol{h}^*$ of Eq. (1) until the relative error satisfies $\|\boldsymbol{h} - \mathcal{F}(\boldsymbol{h}, \boldsymbol{z})\|/\|\boldsymbol{h}\| < 10^{-5}$. Here, the exact gradient is estimated by backpropagation through the fixed-point iterations, and cross-validated by implicit differentiation solved with 20 iterations of the Broyden's method [16]. In our experiment, the cosine similarity between these two gradient estimates consistently succeeds 0.9999, indicating the gradient estimate is quite accurate when the relative error of forward solver is minor. The cosine similarity between phantom gradients and exact gradients is shown in Fig. 1. It shows that the cosine similarity tends to increase as k grows and that a small λ tends to slow down the convergence of the phantom gradient, allowing it to explore in a wider range regarding the angle against the exact gradient.

In the ablation setting, the precision of the phantom gradient during the training process is shown in Fig. 2. The model is trained by implicit differentiation under the official schedule[3]. It shows that the phantom gradient still provides an ascent direction in the real training process, as indicated by the considerable cosine similarity against the exact gradient. Interestingly, the cosine similarity slightly decays as the training progresses, which implies a possibility to construct an adaptive gradient solver for implicit models.

[2]All training sources of this work are available at https://github.com/Gsunshine/phantom_grad.
[3]Code available at https://github.com/locuslab/mdeq.



(a) Neumann-series-based phantom gradient. (b) Unrolling-based phantom gradient.

Figure 2: Cosine similarity between the phantom gradient and the exact gradient in the real scenario. The horizontal axis corresponds to the cosine similarity, and the vertical axis to the training step.

To Pretrain, or not to Pretrain? To better understand the components of the implicit models' training schedule, we first illustrate a detailed ablation study of the baseline model in the ablation setting. The average accuracy with standard deviation is reported in Tab. 1.

The MDEQ model employs a pretraining stage in which the model $\mathcal{F}$ is unrolled as a recurrent network. We study the impact of the pretraining stage, the Dropout [33] operation, and the optimizer separately. It can be seen that the unrolled pretraining stabilizes the training of the MDEQ model. Removing the pretraining stage leads to a severe performance drop and apparent training instability among different trials because the solver cannot obtain an accurate fixed point $\boldsymbol{h}^*$ when the model is not adequately trained. This ablation study also suggests that the MDEQ model is a strong baseline for our method to compare with.

Table 1: Ablation settings on CIFAR-10.

Method	**Acc. (%)**
Implicit Differentiation	85.0 ± 0.2
w/o Pretraining	82.3 ± 1.3
w/o Dropout	83.7 ± 0.1
Adam $\rightarrow$ SGD	84.5 ± 0.3
SGD w/o Pretraining	82.9 ± 1.5
UPG ($\boldsymbol{A}_{5,0.5}$, w/o Dropout)	$\mathbf{85.8} \pm 0.5$
NPG ($\boldsymbol{A}_{5,0.5}$, w/o Dropout)	85.6 ± 0.5
UPG ($\boldsymbol{A}_{9,0.5}$, w/ Dropout)	$\mathbf{86.1} \pm 0.5$

However, pretraining is not always indispensable for training implicit models. It introduces an extra hyperparameter, *i.e.,* how many steps should be involved in the pretraining stage. Next, we discuss how the UPG could circumvent this issue.

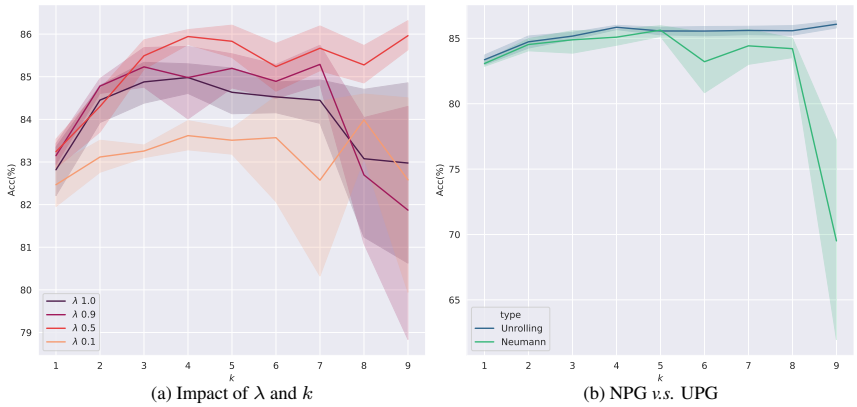
Trade-offs between Unrolling and Neumann. For an exact fixed point $\boldsymbol{h}^*$, *i.e.,* $\boldsymbol{h}^* = \mathcal{F}(\boldsymbol{h}^*, \boldsymbol{z})$, there is no difference between UPG and NPG. However, when the numerical error exists in solving $\boldsymbol{h}^*$, *i.e.,* $\|\boldsymbol{h}^* - \mathcal{F}(\boldsymbol{h}^*, \boldsymbol{z})\| > 0$, these two instatiations of the phantom gradient can behave differently.

We note that a particular benefit of the UPG is its ability to automatically switch between the pretraining and training stages for implicit models. When the model is not sufficiently trained and the solver converges poorly (see [3]), the UPG defines a forward computation graph that is essentially equivalent to a shallow weight-tied network to refine the coarse equilibrium states. In this stage, the phantom gradient serves as a backpropagation through time (BPTT) algorithm and hence behaves as in the pretraining stage. Then, as training progresses, the solver becomes more stable and converges to the fixed point $\boldsymbol{h}^*$ better. This makes the UPG behave more like the NPG. Therefore, the unrolled pretraining is gradually transited into the regular training phase based on implicit differentiation, and the hyperparameter tuning of pretraining steps can be waived. We argue that such an ability to adaptively switch training stages is benign to the implicit models' training protocol, which is also supported by the performance gain in Tab. 1.

Table 2: Complexity comparison. Mem means the memory cost, and K and k denote the solver's steps and the unrolling/Neumann steps, respectively. Here, $K \gg k \approx 1$.

Method	**Time**	**Mem**	**Peak Mem**
Implicit	$\mathcal{O}(K)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$
UPG	$\mathcal{O}(k)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$
NPG	$\mathcal{O}(k)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Although the UPG requires higher memory overhead than implicit differentiation or the NPG, it does not surpass the peak memory usage in the entire training protocol by implicit differentiation



(a) Impact of λ and k (b) NPG *v.s.* UPG

Figure 3: Ablation studies on (a) the hyperparameters λ and k, and (b) two forms of phantom gradient.

due to the pretraining stage. In the ablation setting, the MDEQ model employs a 10-layer unrolling for pretraining, which actually consumes double memory compared with a 5-step unrolling scheme, *e.g.,* $\boldsymbol{A}_{5,0.5}$ in Tab. 1. In Tab. 2, we also demonstrate the time and memory complexity for implicit differentiation and the two forms of phantom gradient.

In addition, leaving the context of approximate and exact gradient aside, we also develop insights into understanding the subtle differences between UPG and NPG in terms of the state-dependent and state-free gradient. Actually, the UPG is state-dependent, which means it corresponds to the "exact" gradient of a computational sub-graph. Both the NPG and the gradient solved by implicit differentiation, however, do not exactly match the gradient of any forward computation graph unless the numerical error is entirely eliminated in both the forward and the backward passes for implicit differentiation or the one-step gradient [34, 21, 30, 22] is used for the NPG, *i.e.,* $k = 1$. Interestingly, we observe that models trained on the state-dependent gradient demonstrate an additional training stability regarding the Jacobian spectral radius, compared with those trained on the state-free counterpart. We empirically note that it can be seen as certain form of implicit Jacobian regularization for implicit models as a supplement to the explicit counterpart [17], indicated by the stable estimated Jacobian spectral radius during training, *i.e.,* $\rho(\partial \mathcal{F}_\lambda / \partial \boldsymbol{h}) \approx 1$.

The experimental results also cohere with our insight. The performance curves in Fig. 3 demonstrate the influence of λ and k and further validate that the UPG is more robust to a wide range of steps k than the NPG. In fact, when the Jacobian spectral radius $\rho(\partial \mathcal{F} / \partial \boldsymbol{h})$ increases freely without proper regularization, the exploding high-order terms in the NPG could exert a negative impact on the overall direction of the phantom gradient, leading to performance degradation when a large k is selected (see Fig. 3(b)). We also observe a surging of the estimated Jacobian spectral radius as well as the gradient explosion issue in the NPG experiments. On the contrary, the UPG can circumvent these problems thanks to its implicit Jacobian regularization.

Table 3: Experiments using DEQ [2] and MDEQ [3] on vision and language tasks. Metrics stand for accuracy(%)↑ for image classification on CIFAR-10 and ImageNet, and perplexity↓ for language modeling on Wikitext-103. JR stands for Jacobian Regularization [17]. † indicates additional steps in the forward equilibrium solver.

Datasets	**Model**	**Method**	**Params**	**Metrics**	**Speed**
CIFAR-10	MDEQ	Implicit	10M	93.8 ± 0.17	$1.0\times$
CIFAR-10	MDEQ	UPG $\boldsymbol{A}_{5,0.5}$	10M	95.0 ± 0.16	$1.4\times$
ImageNet	MDEQ	Implicit	18M	75.3	$1.0\times$
ImageNet	MDEQ	UPG $\boldsymbol{A}_{5,0.6}$	18M	75.7	$1.7\times$
Wikitext-103	DEQ (PostLN)	Implicit	98M	24.0	$1.0\times$
Wikitext-103	DEQ (PostLN)	UPG $\boldsymbol{A}_{5,0.8}$	98M	25.7	$1.7\times$
Wikitext-103	DEQ (PreLN)	JR + Implicit	98M	24.5	$1.7\times$
Wikitext-103	DEQ (PreLN)	JR + UPG $\boldsymbol{A}_{5,0.8}$	98M	24.4	$2.2\times$
Wikitext-103	DEQ (PreLN)	JR + UPG $\boldsymbol{A}_{5,0.8}$	98M	$24.0^\dagger$	$1.7\times$

Phantom Gradient at Scale. We conduct large-scale experiments to verify the advantages of the phantom gradient on vision, graph, and language benchmarks. We adopt the UPG in the large-scale experiments. The results are illustrated in Tab. 3 and Tab. 4. Our method matches or surpasses the implicit differentiation training protocol on the state-of-the-art implicit models with a visible reduction on the training time. When only considering the backward pass, the acceleration for MDEQ can be remarkably $12\times$ on ImageNet classification.

Table 4: Experiments using IGNN [4] on graph tasks. Metrics stand for accuracy(%)↑ for graph classification on COX2 and PROTEINS, Micro-F1(%)↑ for node classification on PPI.

Datasets	Model	Method	Params	Metrics (%)
COX2	IGNN	Implicit	38K	84.1 ± 2.9
COX2	IGNN	UPG $\boldsymbol{A}_{5,0.5}$	38K	83.9 ± 3.0
COX2	IGNN	UPG $\boldsymbol{A}_{5,0.8}$	38K	83.9 ± 2.7
COX2	IGNN	UPG $\boldsymbol{A}_{5,1.0}$	38K	83.0 ± 2.9
PROTEINS	IGNN	Implicit	34K	78.6 ± 4.1
PROTEINS	IGNN	UPG $\boldsymbol{A}_{5,0.5}$	34K	78.4 ± 4.2
PROTEINS	IGNN	UPG $\boldsymbol{A}_{5,0.8}$	34K	78.6 ± 4.2
PROTEINS	IGNN	UPG $\boldsymbol{A}_{5,1.0}$	34K	78.8 ± 4.2
PPI	IGNN	Implicit	4.7M	97.6
PPI	IGNN	UPG $\boldsymbol{A}_{5,0.5}$	4.7M	98.2
PPI	IGNN	UPG $\boldsymbol{A}_{5,0.8}$	4.7M	97.4
PPI	IGNN	UPG $\boldsymbol{A}_{5,1.0}$	4.7M	96.2

4 Related Work

Implicit Models. Implicit models generalize the recursive forward/backward rules of neural networks and characterize their internal mechanism by some pre-specified dynamics. Based on the dynamics, the implicit mechanisms can be broadly categorized into three classes: ODE-based [1, 7], root-solving-based [2, 3, 4, 5, 11], and optimization-based [35, 36, 37] implicit models.

The ODE-based implicit models [1, 7] treat the iterative update rules of residual networks as the Euler discretization of an ODE, which could be solved by any black-box ODE solver. The gradient of the differential equation is calculated using the *adjoint method* [38], in which the adjoint state is obtained by solving another ODE. The root-solving-based implicit models [2, 5, 3, 4, 6, 11, 22] characterize layers of neural networks by solving fixed-point equations. The equations are solved by either the black-box root-finding solver [2, 3] or the fixed-point iteration [4, 22]. The optimization-based implicit models [35, 36, 37, 39, 34, 21, 40, 41] leverage the optimization programs as layers of neural networks. Previous works have studied differentiable layers of quadratic programming [35], submodular optimization [36], maximum satisfiability (MAXSAT) problems [37], and structured decomposition [21]. As for the backward passes, implicit differentiation is applied to the problem-defining equations of the root-solving-based models [2, 3] or the KKT conditions of the optimization-based models [35]. As such, the gradient can be obtained from solving the backward linear system.

In this work, we focus on the root-solving-based implicit models. Theoretical works towards root-solving-based implicit models include the well-posedness [5, 4], monotone operators [11], global convergence [42, 41], and Lipschitz analysis [15]. We look into the theoretical aspect of the gradient-based algorithm in training implicit models and the efficient practice guidance. With these considerations, we show that implicit models of the same architecture could enjoy faster training speed and strong generalization in practical applications by using the phantom gradient.

Non-End-to-End Optimization in Deep Learning. Non-end-to-end optimization aims to replace the standard gradient-based training of deep architectures with modular or weakly modular training without the entire forward and backward passes. Currently, there are mainly three research directions in this field, namely, the auxiliary variable methods [43, 44, 45, 46, 47, 48, 49], target propagation [50, 51, 52], and synthetic gradient [24, 53, 54]. The auxiliary variable methods [43, 44, 45, 46, 47, 48, 49] formulate the optimization of neural networks as constrained optimization problems, in which the layer-wise activations are considered as trainable auxiliary variables. Then, the equality constraints are relaxed as penalty terms added to the objectives so that the parameters and auxiliary variables can be divided into blocks and thus optimized in parallel. The target propagation method [50, 51, 52] trains each module by having its activations regress to the pre-assigned targets, which are

propagated backwards from the downstream modules. Specifically, the auto-encoder architecture is used to reconstruct targets at each layer. Finally, the synthetic gradient method [24, 53, 54] estimates the local gradient of neural networks using auxiliary models, and employ the synthetic gradient in place of the exact gradient to perform parameter update. In this way, the forward and backward passes are decoupled and can be executed in an asynchronous manner.

Our work is in line with the non-end-to-end optimization research since we also aims to decouple the forward and backward passes of neural networks. However, we show that finding a reasonable "target" or a precise gradient estimate is not always necessary in training deep architectures. Our paper paves a path that an inexact but well-conditioned gradient estimate can contribute to both fast training and competitive generalization of implicit models.

Differentiation through Implicit Dynamics. Differentiation through certain implicit dynamics is an important aspect in a wide range of research fields, including bilevel optimization [19, 9], meta-learning [18, 55, 30], and sensitivity analysis [56]. Since the gradient usually cannot be computed analytically, researchers have to implicitly differentiate the dynamics at the converged point. The formula of the gradient typically contains a term of Jacobian-inverse (or Hessian-inverse), which is computationally prohibitive for large-scale models. (See Eq. (2) in our case.) Herein, several techniques have been developed to approximate the matrix inverse in the previous literature.

An intuitive solution is to differentiate through the unrolled steps of a numerical solver of the dynamics [57, 58, 8]. In particular, if a single step is unrolled, it reduces to the well-known *one-step gradient* [59, 18, 60, 34, 30, 21, 22], in which the inverse of Jacobian/Hessian is simply approximated by an identity matrix. On the contrary, unrolling a small number of steps may induce a bias [9], while the memory and computational cost grows linearly as the number of unrolled steps increases. Towards this issue, Shaban *et al.* [19] propose to truncate the long-term dependencies and differentiate through only the last L steps. In fact, if the dynamics have converged to a stationary point, the finite-term truncation in Shaban *et al.* [19] is exactly the Neumann approximation of the Jacobian-inverse with the first L terms. Based on this, Lorraine *et al.* [9] directly use the truncated Neumann series as an approximation of the Jacobian-inverse. Besides the unrolling-based methods, optimization-based approaches [61, 55] have been studied in this field as well. Since the Jacobian-inverse-vector product can be viewed as solution of a linear system, algorithms like the conjugate gradient method can be used to solve it.

5 Limitation and Future Work

The main limitation of this work lies in the hyperparameter tuning of the phantom gradient, especially for the damping factor λ, which directly controls the gradient's precision and conditioning, the implicit Jacobian regularization for UPG, the stability for NPG, and the final generalization behaviors. However, it has not been a hindrance to the application of phantom gradients in training implicit models as one can tune the hyperparameter according to the validation loss in the early training stage.

Regarding future works, we would like to highlight the following aspects: (1) eliminating the bias of the current phantom gradient, (2) constructing an adaptive gradient solver for implicit models, (3) analyzing the damping factor to provide practical guidance, (4) investigating the implicit Jacobian regularization, and (5) understanding how different noises in the gradients can impact the training of implicit models under different loss landscapes.

6 Conclusion

In this work, we explore the possibility of training implicit models via the efficient approximate phantom gradient. We systematically analyze the general condition of a gradient estimate so that the implicit model can be guaranteed to converge to an approximate stationary point of the loss function. Specifically, we give a sufficient condition under which a first-order oracle could always find an ascent direction of the loss landscape in the training process. Moreover, we introduce two instantiations of the proposed phantom gradient, based on either the damped fixed-point unrolling or the Neumann series. The proposed method shows a $1.4 \sim 1.7\times$ acceleration with comparable or better performances on large-scale benchmarks. Overall, this paper provides a practical perspective on training implicit models with theoretical guarantees.

Acknowledgments

Zhouchen Lin was supported by the NSF China (No.s 61625301 and 61731018), NSFC Tianyuan Fund for Mathematics (No. 12026606) and Project 2020BD006 supported by PKU-Baidu Fund. Yisen Wang was partially supported by the National Natural Science Foundation of China under Grant 62006153, and Project 2020BD006 supported by PKU-Baidu Fund. Shaojie Bai was sponsored by a grant from the Bosch Center for Artificial Intelligence.

References

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Neural Information Processing Systems (NeurIPS)*, 2018. 1, 9

[2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models. In *Neural Information Processing Systems (NeurIPS)*, 2019. 1, 2, 3, 6, 8, 9, 23

[3] Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale Deep Equilibrium Models. In *Neural Information Processing Systems (NeurIPS)*, pages 5238–5250, 2020. 1, 2, 6, 7, 8, 9, 22, 23

[4] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit Graph Neural Networks. In *Neural Information Processing Systems (NeurIPS)*, pages 11984–11995, 2020. 1, 2, 9, 23, 24

[5] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. Implicit Deep Learning. *SIAM Journal on Mathematics of Data Science*, 3(3):930–958, 2021. 1, 2, 9

[6] Tiancai Wang, Xiangyu Zhang, and Jian Sun. Implicit Feature Pyramid Network for Object Detection. *arXiv preprint arXiv:2012.13563*, 2020. 1, 9

[7] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. In *Neural Information Processing Systems (NeurIPS)*, 2019. 1, 9

[8] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and Reverse Gradient-Based Hyperparameter Optimization. In *International Conference on Machine Learning (ICML)*, pages 1165–1173, 2017. 1, 10

[9] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing Millions of Hyperparameters by Implicit Differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1540–1552, 2020. 1, 3, 4, 10

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet: Large Scale Visual Recognition Challenge. *International Journal on Computer Vision (IJCV)*, 115(3):211–252, 2015. 1, 2, 6, 23

[11] Ezra Winston and J. Zico Kolter. Monotone operator equilibrium networks. In *Neural Information Processing Systems (NeurIPS)*, pages 10718–10728, 2020. 2, 9

[12] Cheng Lu, Jianfei Chen, Chongxuan Li, Qiuhao Wang, and Jun Zhu. Implicit Normalizing Flows. In *International Conference on Learning Representations (ICLR)*, 2021. 2

[13] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations (ICLR)*, 2017. 2, 6, 23

[14] Chirag Pabbaraju, Ezra Winston, and J. Zico Kolter. Estimating Lipschitz constants of monotone deep equilibrium models. In *International Conference on Learning Representations (ICLR)*, 2021. 2

[15] Max Revay, Ruigang Wang, and Ian R Manchester. Lipschitz Bounded Equilibrium Networks. *arXiv:2010.01732*, 2020. 2, 9

[16] Charles G Broyden. A Class of Methods for Solving Nonlinear Simultaneous Equations. *Mathematics of computation*, 19(92):577–593, 1965. 3, 6

[17] Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Stabilizing Equilibrium Models by Jacobian Regularization. In *International Conference on Machine Learning (ICML)*, 2021. 3, 8, 23, 24

[18] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017. 3, 10

[19] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated Back-propagation for Bilevel Optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1723–1732, 2019. 3, 4, 10

[20] Jens Behrmann, David Kristjanson Duvenaud, and Jörn-Henrik Jacobsen. Invertible Residual Networks. In *International Conference on Machine Learning (ICML)*, 2019. 3

[21] Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. Is Attention Better Than Matrix Decomposition? In *International Conference on Learning Representations (ICLR)*, 2021. 3, 8, 9, 10

[22] Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley J. Osher, and Wotao Yin. Fixed Point Networks: Implicit Depth Models with Jacobian-Free Backprop. *arXiv preprint arXiv:2103.12803*, 2021. 3, 8, 9, 10

[23] Mehrdad Yazdani. Linear Backprop in non-linear networks. In *Neural Information Processing Systems Workshops*, 2018. 3

[24] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled Neural Interfaces using Synthetic Gradients. In *International Conference on Machine Learning (ICML)*, pages 1627–1635, 2017. 3, 9, 10

[25] Xavier Gastaldi. Shake-Shake Regularization. *arXiv preprint arXiv:1705.07485*, 2017. 3

[26] Guozhong An. The Effects of Adding Noise During Backpropagation Training on a Generalization Performance. *Neural Computation*, 8(3):643–674, 1996. 3

[27] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The Anisotropic Noise in Stochastic Gradient Descent: Its Behavior of Escaping from Sharp Minima and Regularization Effects. In *International Conference on Machine Learning (ICML)*, pages 7654–7663, 2019. 3

[28] Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. On the Noisy Gradient Descent that Generalizes as SGD. In *International Conference on Machine Learning (ICML)*, pages 10367–10376, 2020. 3

[29] Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual Flows for Invertible Generative Modeling. *Neural Information Processing Systems (NeurIPS)*, pages 9916–9926, 2019. 3

[30] Xin-Yu Zhang, Taihong Xiao, Haolin Jia, Ming-Ming Cheng, and Ming-Hsuan Yang. Semi-Supervised Learning with Meta-Gradient. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 73–81, 2021. 3, 8, 10

[31] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer, 2009. 6, 22, 23

[32] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2018. 6, 21

[33] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 7

[34] Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. End to end learning and optimization on graphs. In *Neural Information Processing Systems (NeurIPS)*, pages 4672–4683, 2019. 8, 9, 10

[35] Brandon Amos and J. Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *International Conference on Machine Learning (ICML)*, pages 136–145, 2017. 9

[36] Josip Djolonga and Andreas Krause. Differentiable Learning of Submodular Models. *Neural Information Processing Systems (NeurIPS)*, pages 1013–1023, 2017. 9

[37] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning (ICML)*, pages 6545–6554, 2019. 9

[38] VG Boltyanskiy, Revaz V Gamkrelidze, YEF Mishchenko, and LS Pontryagin. Mathematical theory of optimal processes. 1962. 9

[39] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of Blackbox Combinatorial Solvers. In *International Conference on Learning Representations (ICLR)*, 2020. 9

[40] Stephen Gould, Richard Hartley, and Dylan Campbell. Deep Declarative Networks. *IEEE Transactions on Pattern Recognition and Machine Intelligence (PAMI)*, 2021. 9

[41] Xingyu Xie, Qiuhao Wang, Zenan Ling, Xia Li, Yisen Wang, Guangcan Liu, and Zhouchen Lin. Optimization Induced Equilibrium Networks. *arXiv preprint arXiv:2105.13228*, 2021. 9

[42] Kenji Kawaguchi. On the Theory of Implicit Deep Learning: Global Convergence with Implicit Layers. In *International Conference on Learning Representations (ICLR)*, 2020. 9

[43] Miguel Carreira-Perpinan and Weiran Wang. Distributed Optimization of Deeply Nested Systems. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 10–19, 2014. 9

[44] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training Neural Networks Without Gradients: A Scalable ADMM Approach. In *International Conference on Machine Learning (ICML)*, pages 2722–2731, 2016. 9

[45] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient Training of Very Deep Neural Networks for Supervised Hashing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1487–1495, 2016. 9

[46] Ziming Zhang and Matthew Brand. Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks. In *Neural Information Processing Systems (NeurIPS)*, 2017. 9

[47] Jinshan Zeng, Shikang Ouyang, Tim Tsz-Kit Lau, Shaobo Lin, and Yuan Yao. Global Convergence in Deep Learning with Variable Splitting via the Kurdyka-Łojasiewicz Property. *arXiv preprint arXiv:1803.00225*, 2018. 9

[48] Jia Li, Cong Fang, and Zhouchen Lin. Lifted Proximal Operator Machines. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 4181–4188, 2019. 9

[49] Fangda Gu, Armin Askari, and Laurent El Ghaoui. Fenchel Lifted Networks: A Lagrange Relaxation of Neural Network Training. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 3362–3371, 2020. 9

[50] Yoshua Bengio. How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation. *arXiv preprint arXiv:1407.7906*, 2014. 9

[51] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference Target Propagation. In *International Conference on Learning Representations (ICLR)*, pages 498–515, 2015. 9

[52] Alexander Meulemans, Francesco Carzaniga, Johan Suykens, João Sacramento, and Benjamin F. Grewe. A Theoretical Framework for Target Propagation. In *Neural Information Processing Systems (NeurIPS)*, pages 20024–20036, 2020. 9

[53] Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding Synthetic Gradients and Decoupled Neural Interfaces. In *International Conference on Machine Learning (ICML)*, pages 904–912, 2017. 9, 10

[54] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. In *International Conference on Learning Representations (ICLR)*, 2020. 9, 10

[55] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-Learning with Implicit Gradients. In *Neural Information Processing Systems (NeurIPS)*, 2019. 10

[56] J Frédéric Bonnans and Alexander Shapiro. *Perturbation analysis of optimization problems*. Springer Science & Business Media, 2013. 10

[57] Justin Domke. Generic Methods for Optimization-Based Modeling. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 318–326, 2012. 10

[58] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based Hyperparameter Optimization through Reversible Learning. In *International Conference on Machine Learning (ICML)*, pages 2113–2122, 2015. 10

[59] Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters. In *International Conference on Machine Learning (ICML)*, pages 2952–2960, 2016. 10

[60] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. *International Conference on Learning Representations (ICLR)*, 2018. 10

[61] Fabian Pedregosa. Hyperparameter Optimization with Approximate Gradient. In *International Conference on Machine Learning (ICML)*, pages 737–746, 2016. 10

[62] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-performance Deep Learning Library. In *Neural Information Processing Systems (NeurIPS)*, pages 8026–8037, 2019. 16

[63] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, pages 265—283, 2016. 16

[64] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922. 18

[65] Tiberiu Popoviciu. Sur les équations algébriques ayant toutes leurs racines réelles. *Mathematica*, 9:129–145, 1935. 20

A Table of Notions

The notations in this work are summarized in Tab. 5.

Table 5: Table of notations in this work.

Symbol	Description
	Vectors
$\boldsymbol{x}$	Input data
$\boldsymbol{u}$	Injection of input $\boldsymbol{x}$
$\boldsymbol{h}$	Intermediate feature of input $\boldsymbol{x}$
$\hat{\boldsymbol{y}}$	Prediction of input $\boldsymbol{x}$
$\boldsymbol{y}$	Groundtruth of input $\boldsymbol{x}$
$\boldsymbol{\theta}$	Parameter vector of the equilibrium module
$\boldsymbol{z}$	A union of $\boldsymbol{u}$ and $\boldsymbol{\theta}$
	Functions
$\mathcal{M}(\boldsymbol{x})$	Preprocessing module, $\mathcal{M}:\mathbb{R}^{d_{\boldsymbol{x}}}\to\mathbb{R}^{d_{\boldsymbol{u}}}$
$\mathcal{F}(\boldsymbol{h},\,\boldsymbol{z})$	Equilibrium module, $\mathcal{F}:\mathbb{R}^{d_{\boldsymbol{h}}}\times\mathbb{R}^{d_{\boldsymbol{z}}}\to\mathbb{R}^{d_{\boldsymbol{h}}}$
$\mathcal{G}(\boldsymbol{h})$	Postprocessing module, $\mathcal{G}:\mathbb{R}^{d_{\boldsymbol{h}}}\to\mathbb{R}^{d_{\boldsymbol{y}}}$
$\mathcal{R}(\boldsymbol{\theta})$	Loss function, $\mathcal{R}:\mathbb{R}^{d_{\boldsymbol{\theta}}}\to\mathbb{R}$
	Equilibrium states $\boldsymbol{h}$
$\boldsymbol{h}^*$	The equilibrium point of $\mathcal{F}$ given $\boldsymbol{z}$
$\boldsymbol{h}_t$	The intermediate feature of the t^{th} unrolled step
	Gradients & Jacobians
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}$	Exact gradient of the loss *w.r.t.* the parameters $\boldsymbol{\theta}$
$\widehat{\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}}$	Phantom gradient, *i.e.,* an approximation to $\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}$
$\frac{\partial\boldsymbol{a}}{\partial\boldsymbol{b}}$	Gradient of $\boldsymbol{a}$ *w.r.t.* $\boldsymbol{b}$, *i.e.,* $\left(\frac{\partial\boldsymbol{a}}{\partial\boldsymbol{b}}\right)_{ij}=\frac{\partial\boldsymbol{a}_j}{\partial\boldsymbol{b}_i}$.
$\boldsymbol{A}$	An approximation to $\frac{\partial\mathcal{F}}{\partial\boldsymbol{\theta}}\left(\boldsymbol{I}-\frac{\partial\mathcal{F}}{\partial\boldsymbol{h}}\right)^{-1}$
$\boldsymbol{D}$	An approximation to $\left(\boldsymbol{I}-\frac{\partial\mathcal{F}}{\partial\boldsymbol{h}}\right)^{-1}$
	Scalars
$\sigma_{\max},\,\sigma_{\min}$	The maximal/minimal singular value of $\frac{\partial\mathcal{F}}{\partial\boldsymbol{\theta}}$
κ	The condition number of $\frac{\partial\mathcal{F}}{\partial\boldsymbol{\theta}}$
$k,\,\lambda$	The number of steps and damping factor of phantom gradient
$L_{\boldsymbol{h}}$	The Lipschitz constant of $\mathcal{F}$ *w.r.t.* $\boldsymbol{h}$
$d_\diamond$	Dimension of vector $\diamond$
	Operators
$\langle\cdot,\,\cdot\rangle$	Inner product
$\|\cdot\|$	Vector norm or operator norm
$\rho(\cdot)$	Spectral radius

B Algorithm of Phantom Gradient

The following PyTorch-style [62] pseudocode describes the implementation of both the unrolling-based phantom gradient (see Alg. 1) and the Neumann-series-based one (see Alg. 2). To implement the phantom gradient with TensorFlow [63], replace the no_grad context manager with the $stop\_gradient$ operator.

The unrolling-based phantom gradient is computed by the automatic differentiation engine, while the Neumann-series-based phantom gradient is given by Alg. 3. A special reminder is that, for a trained model, removing the unrolling steps in the test stage will not lead to a performance decay but accelerate the inference instead. Similarly, increasing the unrolling steps in the test stage can not further improve the performance, which is validated using MDEQ model on the CIFAR-10 and ImageNet datasets. This implies that the root-finding solver has fully converged to an equilibrium point for the trained model.

Algorithm 1 Unrolling-based phantom gradient, PyTorch-style

```
# solver: the solver to find h*, e.g., the Broyden solver in MDEQ.
# func: the explicit function F that defines the implicit model.
# z: the input variables z to solve h* = F(h*, z)
# h: the solution h* of the implicit module.
# k: the unrolling steps k.
# lambda_: the damping factor λ.
# training: a bool variable that indicates the training or inference stage.

# Forward pass (Backward pass is accomplished by automatic differentiation)
def forward(z, k, lambda_, training):
    with torch.no_grad():
        h = solver(func, z)

    if training:
        for _ in range(k):
            h = (1 - lambda_) * h + lambda_ * func(h, z)

    return h
```

C Proof of Theorems

C.1 Proof of Theorem 1

Proof. Denote $\boldsymbol{J} = \partial \mathcal{F}/\partial \boldsymbol{\theta}$, $\boldsymbol{v} = \partial \mathcal{L}/\partial \boldsymbol{h}$, and $\boldsymbol{u} = \left(\boldsymbol{I} - \partial \mathcal{F}/\partial \boldsymbol{h}\right)^{-1} \boldsymbol{v}$. Let

$$\boldsymbol{E} = \boldsymbol{A}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right) - \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}, \tag{22}$$

and we have $\|\boldsymbol{E}\| < \sigma_{\min}^2/\sigma_{\max}$. Then,

$$\begin{aligned}\left\langle \widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right\rangle &= \boldsymbol{v}^\top \boldsymbol{A}^\top \boldsymbol{J}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^{-1} \boldsymbol{v} = \boldsymbol{u}^\top \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^\top \boldsymbol{A}^\top \boldsymbol{J}\boldsymbol{u} = \boldsymbol{u}^\top \left(\boldsymbol{J} + \boldsymbol{E}\right)^\top \boldsymbol{J}\boldsymbol{u} \\ &\geq \|\boldsymbol{J}\boldsymbol{u}\|^2 - \|\boldsymbol{E}\|\|\boldsymbol{J}\|\|\boldsymbol{u}\|^2 \geq \left(\sigma_{\min}^2 - \sigma_{\max}\|\boldsymbol{E}\|\right)\|\boldsymbol{u}\|^2 > 0,\end{aligned} \tag{23}$$

which concludes the proof. □

Proof of Remerk 1. Suppose $\boldsymbol{A} = \left(\partial \mathcal{F}/\partial \boldsymbol{\theta}\right)\boldsymbol{D}$ and the condition in (8). Then,

$$\left\|\boldsymbol{A}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right) - \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right\| \leq \left\|\frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right\| \left\|\boldsymbol{D}\left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right) - \boldsymbol{I}\right\| < \sigma_{\max} \cdot \frac{1}{\kappa^2} = \frac{\sigma_{\min}^2}{\sigma_{\max}}, \tag{24}$$

indicating the condition in (6) is satisfied. □

Algorithm 2 Neumann-series-based Phantom Gradient, Pytorch-style

```
# solver: the solver to find h*, e.g., the Broyden solver in MDEQ.
# func: the explicit function F that defines the implicit model.
# grad(a, b, c): the function to compute the Jacobian-vector product
    (∂a/∂b) c
# z: the input variables z to solve h* = F(h*, z)
# h: the output h* of the implicit module.
# g: the input gradient ∂L/∂h.
# g_out: the output gradient ∂L/∂z.
# k: the unrolling steps k.
# lambda_: the damping factor λ.

# Forward pass
def forward(z):
    with torch.no_grad():
        h = solver(func, z)

    return h

# Backward pass
def phantom_grad(g, h, z, k, lambda_):
    f = (1 - lambda_) * h + lambda_ * func(h, z)

    g_hat = g
    for _ in range(k-1):
        g_hat = g + grad(f, h, g_hat)

    g_out = lambda_ * grad(f, z, g_hat)
    return g_out
```

Algorithm 3 Neumann-series-based phantom gradient with $\mathcal{O}(1)$ memory

Input $\partial\mathcal{L}/\partial\boldsymbol{h}$, $\mathcal{F}$, $\boldsymbol{h}^*$, k, λ.
Initialize $\hat{\boldsymbol{g}} = \boldsymbol{g} = \partial\mathcal{L}/\partial\boldsymbol{h}$;
$\boldsymbol{f} \leftarrow (1-\lambda)\boldsymbol{h}^* + \lambda\mathcal{F}(\boldsymbol{h}^*, \boldsymbol{z})$
for $i = 1, 2, \cdots, k-1$ **do**
    $\hat{\boldsymbol{g}} \leftarrow \boldsymbol{g} + (\partial\boldsymbol{f}/\partial\boldsymbol{h})\,\hat{\boldsymbol{g}}$; ▷ Compute Jacobian-vector product with automatic differentiation
end for
$\boldsymbol{g}_{\text{out}} \leftarrow \lambda\,(\partial\boldsymbol{f}/\partial\boldsymbol{z})\,\hat{\boldsymbol{g}}$ ▷ Compute Jacobian-vector product to obtain the phantom gradient *w.r.t.* $\boldsymbol{z}$
return $\hat{\boldsymbol{g}}$.

C.2 Proof of Theorem 2

Proof. (i) Since $\|\partial\mathcal{F}/\partial\boldsymbol{h}\| < 1$,

$$\|\boldsymbol{B}\| \leq \lambda \left\|\frac{\partial\mathcal{F}}{\partial\boldsymbol{h}}\right\| + (1-\lambda)\,\|\boldsymbol{I}\| < 1. \tag{25}$$

Let $\boldsymbol{B}_k = \sum_{t=0}^{k-1}\boldsymbol{B}^t$, and for each $p \in \mathbb{N}_+$, we have

$$\|\boldsymbol{B}_{k+p} - \boldsymbol{B}_k\| = \left\|\sum_{t=k}^{k+p-1}\boldsymbol{B}^t\right\| \leq \|\boldsymbol{B}\|^k \left\|\sum_{t=0}^{p-1}\boldsymbol{B}^t\right\| \leq \|\boldsymbol{B}\|^k \sum_{t=0}^{p-1}\|\boldsymbol{B}\|^t < \frac{\|\boldsymbol{B}\|^k}{1-\|\boldsymbol{B}\|}. \tag{26}$$

By the Cauchy's convergence test, the sequence $\{\boldsymbol{B}_k\}$ is convergent. Since

$$(\boldsymbol{I}-\boldsymbol{B})\boldsymbol{B}_k = \boldsymbol{I} - \boldsymbol{B}^k \to \boldsymbol{I}, \quad \text{as } k \to \infty, \tag{27}$$

it follows that $\boldsymbol{B}_k \to (\boldsymbol{I}-\boldsymbol{B})^{-1}$, as $k \to \infty$. Therefore,

$$\lambda \sum_{t=0}^{\infty} \boldsymbol{B}^t = \lambda \left(\boldsymbol{I}-\boldsymbol{B}\right)^{-1} = \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right)^{-1}. \tag{28}$$

(ii) Let $\mathcal{F}_\lambda(\boldsymbol{h}, \boldsymbol{z}) = \lambda \mathcal{F}(\boldsymbol{h}, \boldsymbol{z}) + (1-\lambda)\boldsymbol{h}$, and

$$\frac{\partial \mathcal{F}_\lambda}{\partial \boldsymbol{h}} = \lambda \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}} + (1-\lambda)\boldsymbol{I}. \tag{29}$$

Similar to (25), $\partial \mathcal{F}_\lambda / \partial \boldsymbol{h}$ is also a contraction mapping. By the Banach Fixed Point Theorem [64], the sequence $\{\boldsymbol{h}_t\}$ converges to an exact fixed point $\boldsymbol{h}^*$ of $\mathcal{F}_\lambda$, which is also a fixed point of $\mathcal{F}$.

Denote

$$\boldsymbol{U}_t = \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{h}_t}, \quad \boldsymbol{V}_t = \lambda \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right|_{\boldsymbol{h}_t} + (1-\lambda)\,\boldsymbol{I}. \tag{30}$$

Since the function $\mathcal{F}$ is continuously differentiable *w.r.t.* both $\boldsymbol{h}$ and $\boldsymbol{\theta}$, we have

$$\lim_{t\to\infty} \boldsymbol{U}_t = \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{h}^*} = \boldsymbol{U}_\infty, \quad \lim_{t\to\infty} \boldsymbol{V}_t = \lambda \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right|_{\boldsymbol{h}^*} + (1-\lambda)\,\boldsymbol{I} = \boldsymbol{V}_\infty. \tag{31}$$

According to the conclusion in (i), we have

$$\left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{h}^*} \left(\boldsymbol{I} - \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right|_{\boldsymbol{h}^*}\right)^{-1} = \lambda\, \boldsymbol{U}_\infty \sum_{t=0}^{\infty} \boldsymbol{V}_\infty^t. \tag{32}$$

Comparing Eq. (13) with Eq. (32), we have

$$\begin{aligned}
&\left\| \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{\theta}} - \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{h}^*} \left(\boldsymbol{I} - \left.\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}}\right|_{\boldsymbol{h}^*}\right)^{-1} \right\| = \lambda \left\| \sum_{t=0}^{T-1} \boldsymbol{U}_t \prod_{s=t+1}^{T-1} \boldsymbol{V}_s - \boldsymbol{U}_\infty \sum_{t=0}^{\infty} \boldsymbol{V}_\infty^t \right\| \\
\leq \lambda \Bigg(&\underbrace{\left\| \sum_{t=0}^{T-2} \boldsymbol{U}_t \left(\prod_{s=t+1}^{T-1} \boldsymbol{V}_s - \boldsymbol{V}_\infty^{T-t-1} \right) \right\|}_{\boldsymbol{\Delta}_1} + \underbrace{\left\| \sum_{t=0}^{T-1} \left(\boldsymbol{U}_t - \boldsymbol{U}_\infty\right) \boldsymbol{V}_\infty^{T-t-1} \right\|}_{\boldsymbol{\Delta}_2} + \underbrace{\left\| \boldsymbol{U}_\infty \sum_{t=T}^{\infty} \boldsymbol{V}_\infty^t \right\|}_{\boldsymbol{\Delta}_3} \Bigg).
\end{aligned} \tag{33}$$

In the following context, we prove Eq. (15) by showing that $\boldsymbol{\Delta}_1$, $\boldsymbol{\Delta}_2$, and $\boldsymbol{\Delta}_3$ can be arbitrarily small when T is sufficiently large.

Preparations. For any $\epsilon > 0$, since $\boldsymbol{U}_t \to \boldsymbol{U}_\infty$ and $\boldsymbol{V}_t \to \boldsymbol{V}_\infty$ as $t \to \infty$, there exists $N \in \mathbb{N}_+$ s.t.

$$\|\boldsymbol{U}_t - \boldsymbol{U}_\infty\| < \epsilon, \quad \|\boldsymbol{V}_t - \boldsymbol{V}_\infty\| < \epsilon, \quad \forall t > N. \tag{34}$$

Since $\partial \mathcal{F}_\lambda / \partial \boldsymbol{h}$ is a contraction mapping, there exists $\gamma \in (0,1)$ s.t.

$$\|\boldsymbol{V}_t\| \leq \gamma, \quad \|\boldsymbol{V}_\infty\| \leq \gamma. \tag{35}$$

Besides, since $\partial \mathcal{F} / \partial \boldsymbol{\theta}$ is a continuous function and $\{\boldsymbol{h}_t\}$ is a convergent sequence, it follows that $\{\boldsymbol{h}_t\}$ is contained by a compact set and that $\partial \mathcal{F} / \partial \boldsymbol{\theta}$ is bounded on $\{\boldsymbol{h}_t\}$. Therefore, there exists $M > 0$, s.t.

$$\|\boldsymbol{U}_t\| \leq M, \quad t = 0, 1, 2, \cdots. \tag{36}$$

Taking $t \to \infty$, we have $\|\boldsymbol{U}_\infty\| \leq M$.

For $\boldsymbol{\Delta}_1$. For $t > N$, consider

$$\begin{aligned}
&\left\| \boldsymbol{U}_t \left(\prod_{s=t+1}^{T-1} \boldsymbol{V}_s - \boldsymbol{V}_\infty^{T-t-1} \right) \right\| \\
\leq &\|\boldsymbol{U}_t\| \sum_{s=t+1}^{T-1} \left\| \boldsymbol{V}_{t+1}\boldsymbol{V}_{t+2}\cdots\boldsymbol{V}_s\boldsymbol{V}_\infty^{T-s-1} - \boldsymbol{V}_{t+1}\boldsymbol{V}_{t+2}\cdots\boldsymbol{V}_{s-1}\boldsymbol{V}_\infty^{T-s} \right\| \\
\leq &\|\boldsymbol{U}_t\| \sum_{s=t+1}^{T-1} \|\boldsymbol{V}_{t+1}\| \|\boldsymbol{V}_{t+2}\| \cdots \|\boldsymbol{V}_{s-1}\| \|\boldsymbol{V}_s - \boldsymbol{V}_\infty\| \|\boldsymbol{V}_\infty\|^{T-s-1} \\
\leq &M(T-t-1)\gamma^{T-t-2}\epsilon,
\end{aligned} \tag{37}$$

and for $t \leq N$, we simply have

$$\left\| \boldsymbol{U}_t \left(\prod_{s=t+1}^{T-1} \boldsymbol{V}_s - \boldsymbol{V}_\infty^{T-t-1} \right) \right\| \leq \|\boldsymbol{U}_t\| \left(\prod_{s=t+1}^{T-1} \|\boldsymbol{V}_s\| + \|\boldsymbol{V}_\infty\|^{T-t-1} \right) \leq 2M\gamma^{T-t-1}. \tag{38}$$

Therefore, when $T > N+2$, $\boldsymbol{\Delta}_1$ can be bounded as follows:

$$\begin{aligned} \boldsymbol{\Delta}_1 &\leq \left(\sum_{t=0}^{N} + \sum_{t=N+1}^{T-2} \right) \left\| \boldsymbol{U}_t \left(\prod_{s=t+1}^{T-1} \boldsymbol{V}_s - \boldsymbol{V}_\infty^{T-t-1} \right) \right\| \\ &\leq 2M \sum_{t=0}^{N} \gamma^{T-t-1} + M\epsilon \sum_{t=N+1}^{T-2} (T-t-1)\gamma^{T-t-2} \\ &\leq 2M\gamma^{T-N-1} \frac{1-\gamma^{N+1}}{1-\gamma} + \left(\frac{1-\gamma^{T-N-2}}{(1-\gamma)^2} - \frac{(T-N-2)\gamma^{T-N-2}}{1-\gamma} \right) M\epsilon \\ &\leq \frac{2M}{1-\gamma}\gamma^{T-N-1} + \frac{M}{(1-\gamma)^2}\epsilon. \end{aligned} \tag{39}$$

Since $M/(1-\gamma)^2$ is a constant and $\gamma^{T-N-1} \to 0$ as $T \to \infty$, $\boldsymbol{\Delta}_1$ can be arbitrarily small for a sufficiently large T.

For $\boldsymbol{\Delta}_2$. Consider

$$\left\| \left(\boldsymbol{U}_t - \boldsymbol{U}_\infty \right) \boldsymbol{V}_\infty^{T-t-1} \right\| \leq \|\boldsymbol{U}_t - \boldsymbol{U}_\infty\| \, \|\boldsymbol{V}_\infty\|^{T-t-1} \leq \begin{cases} \gamma^{T-t-1}\epsilon, & \text{when } t \geq N; \\ 2M\gamma^{T-t-1} & \text{when } t < N. \end{cases} \tag{40}$$

Therefore, when $T > N+2$, $\boldsymbol{\Delta}_2$ can be bounded as follows:

$$\begin{aligned} \boldsymbol{\Delta}_2 &\leq \left(\sum_{t=0}^{N} + \sum_{t=N+1}^{T-1} \right) \left\| \left(\boldsymbol{U}_t - \boldsymbol{U}_\infty \right) \boldsymbol{V}_\infty^{T-t-1} \right\| \leq 2M \sum_{t=0}^{N} \gamma^{T-t-1} + \epsilon \sum_{t=N+1}^{T-1} \gamma^{T-t-1} \\ &\leq \frac{2M}{1-\gamma}\gamma^{T-N-1} + \frac{\epsilon}{1-\gamma}. \end{aligned} \tag{41}$$

Since $1/(1-\gamma)$ is a constant and $\gamma^{T-N-1} \to 0$ as $T \to \infty$, $\boldsymbol{\Delta}_2$ can be arbitrarily small for a sufficiently large T.

For $\boldsymbol{\Delta}_3$. As $t \to \infty$, we have

$$\left\| \boldsymbol{U}_\infty \sum_{t=T}^{\infty} \boldsymbol{V}_\infty^t \right\| \leq \|\boldsymbol{U}_\infty\| \, \|\boldsymbol{V}_\infty\|^T \left\| \left(\boldsymbol{I} - \boldsymbol{V}_\infty \right)^{-1} \right\| \leq M \cdot \gamma^T \cdot \frac{1}{1-\gamma} \to 0. \tag{42}$$

As a result, we obtain the conclusion in Eq. (15). □

C.3 Proof of Theorem 3

Proof. Let $\widehat{\frac{\partial \mathcal{L}_n}{\partial \boldsymbol{\theta}}}$ be the phantom gradient at the n^{th} iteration. By ℓ-smoothness of $\mathcal{R}$, we have

$$\begin{aligned} \mathcal{R}(\boldsymbol{\theta}_{n+1}) &\leq \mathcal{R}(\boldsymbol{\theta}_n) + \langle \nabla \mathcal{R}(\boldsymbol{\theta}_n), \boldsymbol{\theta}_{n+1} - \boldsymbol{\theta}_n \rangle + \frac{\ell}{2} \|\boldsymbol{\theta}_{n+1} - \boldsymbol{\theta}_n\|^2 \\ &= \mathcal{R}(\boldsymbol{\theta}_n) - \eta_n \left\langle \nabla \mathcal{R}(\boldsymbol{\theta}_n), \widehat{\frac{\partial \mathcal{L}_n}{\partial \boldsymbol{\theta}}} \right\rangle + \frac{\ell \eta_n^2}{2} \left\| \widehat{\frac{\partial \mathcal{L}_n}{\partial \boldsymbol{\theta}}} \right\|^2. \end{aligned} \tag{43}$$

Let

$$\boldsymbol{e}_n = \left. \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n} - \widehat{\frac{\partial \mathcal{L}_n}{\partial \boldsymbol{\theta}}} \tag{44}$$

be the approximation error at the n^{th} iteration. Taking expectation *w.r.t.* the first n iterations, we have

$$\mathbb{E}_{1\sim n} \left[\mathcal{R}(\boldsymbol{\theta}_{n+1}) \right] = \mathbb{E}_{1\sim n-1} \left[\mathbb{E}_n \left[\mathcal{R}(\boldsymbol{\theta}_{n+1}) \,|\, 1 \sim n-1 \right] \right] = \mathbb{E}_{1\sim n-1} \left[\mathbb{E}_n \left[\mathcal{R}(\boldsymbol{\theta}_{n+1}) \,|\, \boldsymbol{\theta}_n \right] \right], \tag{45}$$

where the first equality comes from the *law of total expectation*, while the second from the fact that the stochasticity of the first $n-1$ steps is totally captured by the value $\boldsymbol{\theta}_n$. Consider the inner expectation in Eq. (45), and we omit the condition on $\boldsymbol{\theta}_n$ when no ambiguity is made. Note that in the following derivation, all expectations, variances, and covariances are conditioned on $\boldsymbol{\theta}_n$.

$$\begin{aligned}\mathbb{E}_n\left[\mathcal{R}(\boldsymbol{\theta}_{n+1})\right] &\leq \mathbb{E}_n\left[\mathcal{R}(\boldsymbol{\theta}_n) - \eta_n\left\langle\nabla\mathcal{R}(\boldsymbol{\theta}_n), \widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right\rangle + \frac{\ell\eta_n^2}{2}\left\|\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right\|^2\right] \\ &= \mathcal{R}(\boldsymbol{\theta}_n) - \eta_n\left\langle\nabla\mathcal{R}(\boldsymbol{\theta}_n), \mathbb{E}_n\left[\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right]\right\rangle + \frac{\ell\eta_n^2}{2}\mathbb{E}_n\left[\left\|\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right\|^2\right],\end{aligned} \tag{46}$$

where

$$\mathbb{E}_n\left[\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right] = \mathbb{E}_n\left[\left.\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n} - \boldsymbol{e}_n\right] = \nabla\mathcal{R}(\boldsymbol{\theta}_n) - \mathbb{E}_n\left[\boldsymbol{e}_n\right], \tag{47}$$

and

$$\mathbb{E}_n\left[\left\|\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right\|^2\right] = \left\|\mathbb{E}_n\left[\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right]\right\|^2 + \operatorname{tr}\left(\operatorname{Cov}_n\left(\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right)\right). \tag{48}$$

Suppose $\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\| \leq G$ almost surely, and then we have

$$\left\|\mathbb{E}_n\left[\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right]\right\|^2 = \left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n) - \mathbb{E}_n\left[\boldsymbol{e}_n\right]\right\|^2 \leq (G+\epsilon)^2. \tag{49}$$

Moreover, by the properties of covariance,

$$\begin{aligned}&\operatorname{tr}\left(\operatorname{Cov}_n\left(\widehat{\frac{\partial\mathcal{L}_n}{\partial\boldsymbol{\theta}}}\right)\right) = \operatorname{tr}\left(\operatorname{Cov}_n\left(\left.\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n} - \boldsymbol{e}_n\right)\right) \\ &= \operatorname{tr}\left(\operatorname{Cov}_n\left(\left.\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n}\right)\right) + \operatorname{tr}\left(\operatorname{Cov}_n\left(\boldsymbol{e}_n\right)\right) - 2\operatorname{tr}\left(\operatorname{Cov}_n\left(\left.\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n}, \boldsymbol{e}_n\right)\right) \\ &\leq 2\operatorname{tr}\left(\operatorname{Cov}_n\left(\left.\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n}\right)\right) + 2\operatorname{tr}\left(\operatorname{Cov}_n\left(\boldsymbol{e}_n\right)\right),\end{aligned} \tag{50}$$

where the last inequility comes from

$$\begin{aligned}\left|\operatorname{tr}\left(\operatorname{Cov}\left(\boldsymbol{a}, \boldsymbol{b}\right)\right)\right| &\leq \sum_i \left|\operatorname{Cov}\left(a_i, b_i\right)\right| \leq \sum_i \sqrt{\operatorname{Var}\left(a_i\right)\operatorname{Var}\left(b_i\right)} \leq \sum_i \frac{\operatorname{Var}\left(a_i\right) + \operatorname{Var}\left(b_i\right)}{2} \\ &= \frac{1}{2}\left(\operatorname{tr}\left(\operatorname{Cov}\left(\boldsymbol{a}\right)\right) + \operatorname{tr}\left(\operatorname{Cov}\left(\boldsymbol{b}\right)\right)\right).\end{aligned} \tag{51}$$

By the Popoviciu's inequality on variances [65], the second term in (50) can be bounded by $d_{\boldsymbol{\theta}}\epsilon^2$, *i.e.,*

$$\operatorname{tr}\left(\operatorname{Cov}_n\left(\boldsymbol{e}_n\right)\right) \leq d_{\boldsymbol{\theta}}\epsilon^2, \tag{52}$$

where $d_{\boldsymbol{\theta}}$ denotes the dimension of $\boldsymbol{\theta}$. Finally, since the gradient estimator $\partial\mathcal{L}/\partial\boldsymbol{\theta}$ has a bounded covariance, there exists $M > 0$, s.t.

$$\operatorname{tr}\left(\operatorname{Cov}_n\left(\left.\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_n}\right)\right) \leq M, \quad \text{almost surely.} \tag{53}$$

Combining (46), (47), (48), (49), (50), and (53), we have

$$\begin{aligned}\mathbb{E}_n\left[\mathcal{R}(\boldsymbol{\theta}_{n+1})\right] &\leq \mathcal{R}(\boldsymbol{\theta}_n) - \eta_n\left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\right\|^2 + \eta_n\left\langle\nabla\mathcal{R}(\boldsymbol{\theta}_n), \mathbb{E}_n\left[\boldsymbol{e}_n\right]\right\rangle + K\eta_n^2, \\ &\leq \mathcal{R}(\boldsymbol{\theta}_n) - \eta_n\left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\right\|^2 + \eta_n\left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\right\|\left\|\mathbb{E}_n\left[\boldsymbol{e}_n\right]\right\| + K\eta_n^2 \\ &\leq \mathcal{R}(\boldsymbol{\theta}_n) - \eta_n\left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\right\|^2 + \eta_n G\epsilon + K\eta_n^2,\end{aligned} \tag{54}$$

where $K = \ell\left((G+\epsilon)^2 + 2M + 2d_{\boldsymbol{\theta}}\epsilon^2\right)/2$ is a constant. Substitute (54) into Eq. (45), and it becomes

$$\mathbb{E}_{1\sim n}\left[\mathcal{R}(\boldsymbol{\theta}_{n+1})\right] \leq \mathbb{E}_{1\sim n-1}\left[\mathcal{R}(\boldsymbol{\theta}_n)\right] - \eta_n \mathbb{E}_{1\sim n-1}\left[\left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\right\|^2\right] + \eta_n G\epsilon + K\eta_n^2. \tag{55}$$

By taking a summation over the first N steps, we have

$$\begin{aligned}\mathbb{E}_{1\sim N}\left[\sum_{n=1}^{N}\eta_n\left\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\right\|^2\right] &\leq \mathcal{R}(\boldsymbol{\theta}_1) - \mathbb{E}_{1\sim N}\left[\mathcal{R}(\boldsymbol{\theta}_{N+1})\right] + G\epsilon\sum_{n=1}^{N}\eta_n + K\sum_{n=1}^{N}\eta_n^2 \\ &\leq \mathcal{R}(\boldsymbol{\theta}_1) - m + G\epsilon\sum_{n=1}^{N}\eta_n + K\sum_{n=1}^{N}\eta_n^2,\end{aligned} \tag{56}$$

where $m = \inf_{\boldsymbol{\theta}}\mathcal{R}(\boldsymbol{\theta})$ since $\mathcal{R}$ is lower-bounded. Dividing a factor of $\sum_{n=1}^{N}\eta_n$, we have

$$\mathbb{E}_{1\sim N}\left[\frac{\sum_{n=1}^{N}\eta_n\|\nabla\mathcal{R}(\boldsymbol{\theta}_n)\|^2}{\sum_{n=1}^{N}\eta_n}\right] \leq G\epsilon + \frac{\mathcal{R}(\boldsymbol{\theta}_1) - m}{\sum_{n=1}^{N}\eta_n} + K\frac{\sum_{n=1}^{N}\eta_n^2}{\sum_{n=1}^{N}\eta_n}. \tag{57}$$

Since $\eta_n = \mathcal{O}(1/\sqrt{n})$, it follows that

$$\sum_{n=1}^{N}\eta_n = \mathcal{O}\left(\sqrt{N}\right), \quad \frac{\sum_{n=1}^{N}\eta_n^2}{\sum_{n=1}^{N}\eta_n} = \mathcal{O}\left(\frac{\log N}{\sqrt{N}}\right). \tag{58}$$

Combining (57) and Eq. (58) concludes the proof. □

Remark 3. The assumption that $\mathcal{R}$ has almost-surely bounded gradient at $\{\boldsymbol{\theta}_n\}_{n=0}^{N}$ is reasonable. Because of the existence of norm-based regularizations, *e.g.,* weight decay, we can assume $\boldsymbol{\theta}$ is almost surely optimized within a compact set in the parameter space. If we further assume $\mathcal{R}$ is continuously differentiable, the almost-sure boundedness of $\|\nabla\mathcal{R}\|$ within the compact set follows its continuity.

Remark 4. We justify the assumption that the gradient in Eq. (4) has a bounded covariance. For the SGD algorithm, the stochasticity of the gradient in Eq. (4) comes from the random sampling of the training example (or the training mini-batch) from the dataset. Since there are finite samples in the training set, the covariance of Eq. (4) remains finite. Moreover, as Theorem 2 only considers a finite training schedule, *i.e.,* N steps, the possible combination of the selected sample (or mini-batch) at each step is still finite (even though its number grows combinatorially). Therefore, it is reasonable to assume the gradient in Eq. (4) has a bounded covariance.

D Experiment Details

In this section, we introduce the experimental settings of this paper in detail and discuss some additional findings of training implicit models.

D.1 Synthetic Setting

For the synthetic setting, the following model is used:

$$\boldsymbol{h}^* = \mathcal{F}\left(\boldsymbol{h}^* + \boldsymbol{u}\right) \tag{59}$$

where $\mathcal{F}$ is an 1-layer network with spectral normalization [32], and $\boldsymbol{u}$, $\boldsymbol{h}^* \in \mathbb{R}^{N\times D}$. The loss $\mathcal{L}$ is given by the mean squared error (MSE) between $\boldsymbol{h}^*$ and $\boldsymbol{y}$. We choose $N = 32$ and $D = 128$ and randomly sample 50000 data pairs $(\boldsymbol{u},\ \boldsymbol{y})$ to compute the gradient $\partial\mathcal{L}/\partial\boldsymbol{u}$.

We generate a symmetric weight matrix for the network and constrain the Lipschitz constant $L_{\boldsymbol{h}}$ to a given level using spectral normalization. For the visualization in the main paper, we adopt $L_{\boldsymbol{h}} = 0.9$. For the additional visualization on the stability of the solver in Fig. 4, we choose $L_{\boldsymbol{h}}$ from $\{0.9,\ 0.99,\ 0.999,\ 0.9999\}$.

To solve $\boldsymbol{h}^*$, we employ the fixed-point iteration as the solver. For the synthetic setting, we use 100 fixed-point iterations to obtain $\boldsymbol{h}^*$ that satisfies the relative error $\|\boldsymbol{h}^* - \mathcal{F}(\boldsymbol{h}^*, \boldsymbol{u})\|/\|\boldsymbol{h}^*\| \leq 10^{-5}$. For the visualization in Fig. 4, we also apply 100 fixed-point iterations for each $L_{\boldsymbol{h}}$.

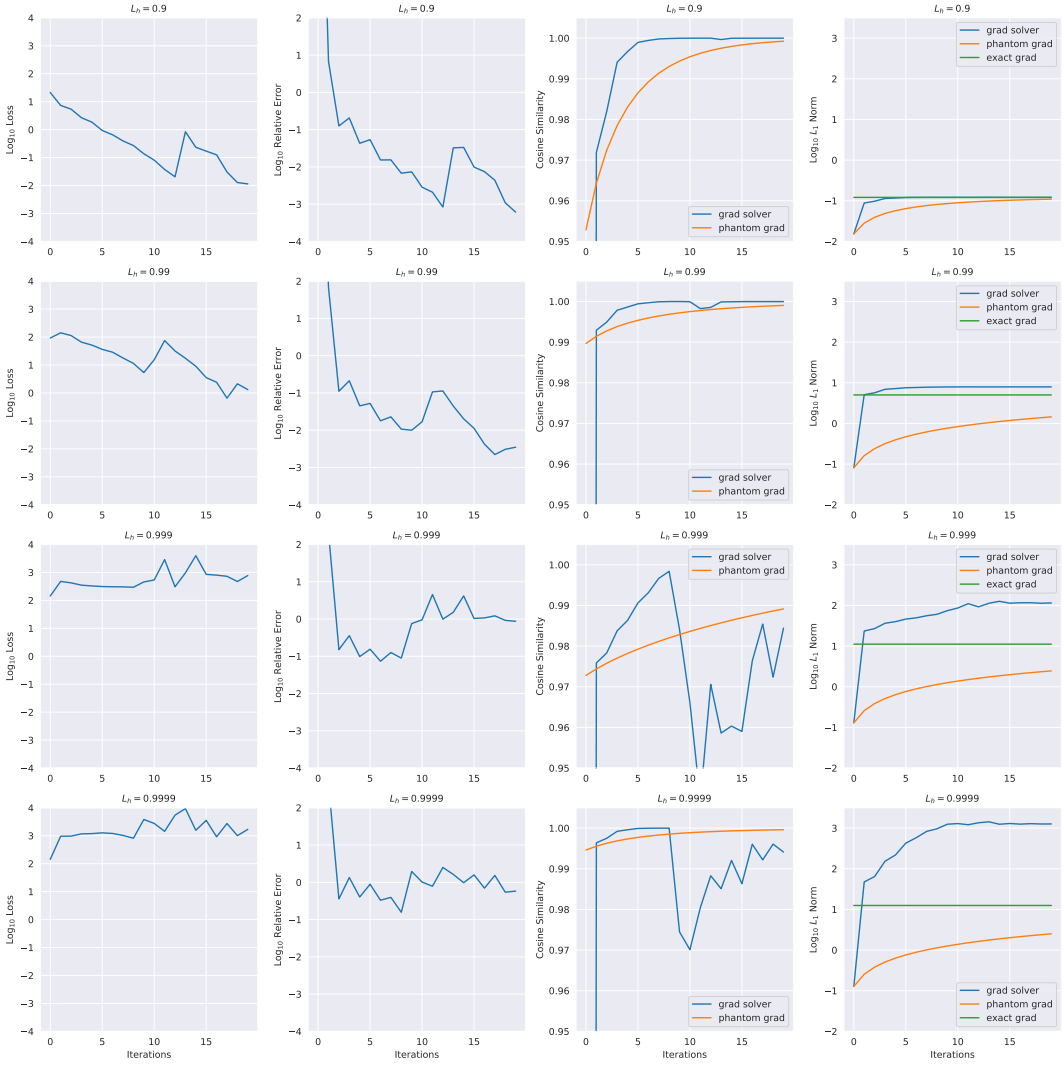


Figure 4: Visualization of gradient solvers under different $L_{\boldsymbol{h}}$.

D.2 Ablation Setting

For the ablation setting, we use the original MDEQ-Tiny [3] model (170K parameters) on CIFAR-10 [31] classification without any architecture modification. Therefore, the performance gain upon the state-of-the-art method is due to the improved training efficiency thanks to the proposed phantom gradient.

The experiments are conducted without data augmentation as in [3]. The training schedule, batch size, cosine learning rate annealing strategy, and other hyperparameters are kept unchanged for all ablation experiments. We also follow the official training protocol of MDEQ[4] to reproduce its result.

For the training protocol without pretraining, we substitute the unrolled pretraining stage by implicit differentiation. For the training protocol without Dropout, we remove the variational Dropout from the model. We also experiment with the SGD optimizer under the standard hyperparameter setting, *i.e.,* a learning rate of 0.1, a momentum of 0.9, and a weight decay of 0.0001.

We train the MDEQ model using the two types of phantom gradient with the SGD optimizer (under the hyperparameters mentioned above) and other hyperparameters unchanged from the original setting. The model is trained without shallow-layer pretraining, suggesting an $\mathcal{O}(k)$ and $\mathcal{O}(1)$ peak memory

[4]Code available at https://github.com/locuslab/mdeq.

usage for the unrolling-based and the Neumann-series-based phantom gradient, respectively. In both cases, the damped fixed-point iteration starts at the solution obtained by the Broyden's method.

We monitor the Jacobian spectral radius $\rho(\partial \mathcal{F}/\partial \boldsymbol{h})$ during training for both forms of phantom gradient. It shows that the radius can grow without restriction for the state-free NPG when the phantom gradient includes high-order terms and cannot exactly match the gradient of a computational sub-graph. A similar phenomenon is observed when using the state-free gradient estimate from implicit differentiation with considerable numerical errors in the forward and backward passes [17]. On the contrary, for the state-dependent UPG, the Jacobian spectral radius is kept within a reasonable region during training thanks to the implicit Jacobian regularization.

D.3 Experiments at Scale

For large-scale experiments, we adopt MDEQ and MDEQ-Small on the CIFAR-10 [31] and ImageNet [10] benchmarks, respectively, DEQ (PostLN) [2] and DEQ (PreLN) [17] on the Wikitext-103 [13] dataset, and IGNN [4] on graph classification (COX2, PROTEINS) and node classification (PPI) benchmarks.

ConvNet-based Implicit Models on Vision Datasets. To train MDEQ on CIFAR-10, we employ the UPG with $\lambda = 0.5$ and $k = 5$, *i.e.,* $\boldsymbol{A}_{5,0.5}$. Besides, we use the SGD optimizer with a learning rate of 0.1, a momentum of 0.9, and a weight decay of 0.0001, and keep other experimental settings unchanged, including the number of training epochs, the batch size, the learning rate annealing strategy, *etc.*

We adopt two settings on ImageNet. The first setting follows the practice of [3] to pretrain the model for the same number of epochs. Afterwards, the UPG with $\boldsymbol{A}_{5,0.6}$ is used to train the model for the remaining training schedule. This setting achieves a test accuracy of 75.2%. In the second setting, we adopt the UPG to train the implicit model throughout, leaving the UPG to automatically transit from the pretraining stage to the regular training stage. This setting demonstrates a test accuracy of 75.7%. The difference confirms that the automatic transition property of UPG helps alleviate the burden of hyperparameter tuning, *i.e.,* the number of steps in the pretraining stage, and benefit to the final performance as well.

We also verify the implicit Jacobian regularization from the UPG on ImageNet. By calculating the Jacobian spectral radius of the trained model on the validation set through the power method, we find that the radius $\rho(\partial \mathcal{F}_\lambda/\partial \boldsymbol{h})$ is retained around 1, although the radius of the equilibrium module $\rho(\partial \mathcal{F}/\partial \boldsymbol{h})$ usually exceeds 1 (but also remains bounded). This finding provides us with a potential path to explain why the damping operation can enhance the naive unrolling to match or even surpass the standard implicit differentiation for ConvNets on vision tasks. We conjecture that the damping operation allows the equilibrium module to evolve within a wider range, *e.g.,* $\rho(\partial \mathcal{F}/\partial \boldsymbol{h}) > 1$, which contributes to its better representative capacity, while maintaining stability regarding the backward pass, *i.e.,* $\rho(\partial \mathcal{F}_\lambda/\partial \boldsymbol{h}) \approx 1$.

Transformer-based Implicit Models on Language Datasets. For language modeling on Wikitext-103, we follow the official training protocol of the DEQ model [2]. However, the UPG leads to inferior generalization capacity on the test set while the training loss is similar to that of implicit differentiation. The NPG even fails to optimize the DEQ (PostLN) model unless the explicit Jacobian regularization [17] or the adaptive damping factor, *e.g.,* $\lambda = 1/\rho(\partial \mathcal{F}/\partial \boldsymbol{h})$, is applied.

The performance discrepancy of different implicit models suggests the following perspective. The loss landscape and training strategy are the two sides of the same coin. Architecture, dataset, and loss function jointly define the loss landscape that has considerable impact on the preferable training strategy. For the ConvNet-based implicit model trained on vision tasks, the loss landscape is likely more regular so that the model trained on the phantom gradient can extricate itself from severe overfitting and achieve remarkable performance with acceleration despite the biased gradient estimate (which means the approximation error cannot be easily zeroed out by taking the expectation over the data distribution). For the Transformer-based implicit model on language processing tasks, in contrast, it is more arduous to employ the phantom gradient due to a lack of regularity of the loss landscape, thus inspiring us to supplement with additional regularization on the loss landscape.

To this end, we introduce the explicit Jacobian regularization (JR) [17] to strengthen the regularity of the loss landscape. The training protocol follows the official source of DEQ with JR[5]. Note that with the implicit Jacobian regularization effect of the UPG, the weight of the explicit JR can be significantly reduced, *e.g.,* from 2.0 to 0.1, and the training stability is still maintained. Meanwhile, the explicit JR can also play a vital role in alleviating overfitting for the UPG. Combining the UPG with explicit JR demonstrates an impressive test perplexity of 24.4 with $2.2\times$ training acceleration (with 14 forward Broyden iterations), and a test perplexity of 24.0 with $1.7\times$ training acceleration (with 20 forward Broyden iterations).

Our results indicate that it is more tactful to understand the training strategy combined with the loss landscape instead of only focusing on the former but neglecting the latter.

GNN-based Implicit Models on Graph Datasets. To conduct experiments on graph datasets, we follow the default architectures and training settings of the IGNN model [4][6]. We employ different damping factor λ for both graph classification and node classification. In the experiments, we encounter the training stability issue for IGNN on the PPI node classification task. Specifically, the IGNN model suffers from training collapse when using either the UPG or the exact gradient by implicit differentiation. Hence the best result from three runs is reported for this task. We conjecture that the stability issue comes from hyperparameter selection regarding the projected gradient in IGNN, since it is not easy to figure out the proper hyperparameters for well-posedness. For graph classification, the stability issue is not observed.

D.4 Additional Analysis on the Gradient Solver

To illustrate the vulnerability the gradient solver for implicit differentiation in the ill-conditioned cases, we provide the optimization dynamics in Fig. 4 and its comparison with the phantom gradient in the synthetic setting. We plot (1) the optimization objective $\|(I - \partial \mathcal{F}/\partial \boldsymbol{h})\hat{\boldsymbol{g}} - \partial \mathcal{L}/\partial \boldsymbol{h}\|$, (2) the relative error $\|(I - \partial \mathcal{F}/\partial \boldsymbol{h})\hat{\boldsymbol{g}} - \partial \mathcal{L}/\partial \boldsymbol{h}\|/\|\hat{\boldsymbol{g}}\|$, (3) the cosine similarity between the solved gradient $\hat{\boldsymbol{g}}$ (or the phantom gradient) and the exact gradient $\boldsymbol{g}$, and (4) the L_1 norm of the solved gradient $\hat{\boldsymbol{g}}$, the phantom gradient, and the exact gradient $\boldsymbol{g}$. Here, in the context of optimization, $\hat{\boldsymbol{g}}$ is the solution of the backward linear system solved by the Broyden's method.

Fig. 4 shows that the gradient solver diverges in ill-conditioned situations. It is shown that the phantom gradient demonstrates much better stability, especially in the extremely ill-conditioned cases, *e.g.,* $L_{\boldsymbol{h}} = 0.9999$. As for the Broyden's method, more optimization steps do not necessarily make the solved gradient more aligned to the exact gradient, as indicated by the oscillating cosine similarity. Besides, the norm of the solved gradient also tends to explode in the optimization process, while the phantom gradient maintains a moderate norm throughout.

[5]Code available at https://github.com/locuslab/deq.

[6]Code available at https://github.com/SwiftieH/IGNN.