On Training Implicit Models

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

[1]Peking University, [2]Carnegie Mellon University, [3]Pazhou Lab

Background: Implicit Models

- DEQ-style Implicit Models [1]
  - Given a union $\boldsymbol{z}$ of the parameters $\boldsymbol{\theta}$ and the injection $\boldsymbol{u} = \mathcal{M}(\boldsymbol{x})$ from the input data $\boldsymbol{x}$
  - The output of the equilibrium module $\mathcal{F}$ is defined as the equilibrium point $\boldsymbol{h}^*$ of the dynamics,

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

  - Post-processing module $\mathcal{G}$: $\widehat{\boldsymbol{y}} = \mathcal{G}(\boldsymbol{h}^*)$ and Loss function $\mathcal{L}$, *etc*.

- Implicit Differentiation
  - Differentiate the dynamics via Implicit Function Theorem (IFT).

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

[1] Shaojie Bai, J. Zico Kolter, Vladlen Koltun. Deep Equilibrium Models

Motivation

- Implicit Differentiation
  - Differentiate the dynamics via Implicit Function Theorem (IFT).

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{h}^*}{\partial \boldsymbol{\theta}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^*} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}} \right)^{-1} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^*}$$

- Motivation:
  - 1. Expense cost for the inverse in the exact gradient, e.g., $\partial \mathcal{F} / \partial \boldsymbol{h}$ is of $10^6 \times 10^6$ size.
  - 2. The conditioning issue and the numerical stability.
  - 3. Moderate gradient noise can help generalization.

Key Ideas

- We calculate the exact but expensive gradient via IFT.

- Our target?
  - Calculate the (exact but expensive) gradient? No. --------- Method
  - Train the implicit models? Yes! --------- Target

- Gradient noise is acceptable for the optimization purpose.
  - SGD is naturally noisy!

- Phantom Gradient: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{h}^*}{\partial \boldsymbol{\theta}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^*} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} \left(\boldsymbol{I} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{h}} \right)^{-1} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}^*}$ → $\widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}} \triangleq \boldsymbol{A} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$

General Condition

$$\widehat{\frac{\partial \mathcal{L}}{\partial \theta}} := \boldsymbol{A} \frac{\partial \mathcal{L}}{\partial h} \qquad \left\langle \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{x}}} \right\rangle > 0$$

- **Theorem 1**. Let $\sigma_{\max}$ and $\sigma_{\min}$ be the maximal and minimal singular values of $\partial\mathcal{F}/\partial\boldsymbol{\theta}$. If

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

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

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

Instantiations

$$\widehat{\frac{\partial \mathcal{L}}{\partial \theta}} := \boldsymbol{A} \frac{\partial \mathcal{L}}{\partial h} \qquad \left\langle \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \widehat{\frac{\partial \mathcal{L}}{\partial \mathbf{x}}} \right\rangle > 0$$

- Unrolling-based Phantom Grad (**UPG**)
  - Considering the the damped fixed-point iteration,

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

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

- Neumann-series-based Phantom Grad (**NPG**)
  - Considering the the damped fixed-point iteration,

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

Pseudo Code for UPG

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 models.
# training: a bool variable that indicates training or inference.
# k: the unrolling step k.
# lambda_: the damping factor λ.

# a plain forward pass using Pytorch
# calculate the phantom gradient by automatic differentiation
# input: z & output: h
def forward(z):
    with torch.no_grad():
        h = solver(func, z)

    # define the computational graph for the backward pass.
    # only used in the training stage
    if training:
        for _ in range(k):
            h = (1 - lambda_) * h + lambda_ * func(h, z)

    return h
```

Pseudo Code for NPG

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 model.
# k: the unrolling step k.
# lambda_: the damping factor λ.

# a plain forward pass using Pytorch
# input: z & output: h
def forward(z):
    with torch.no_grad():
        h = solver(func, z)

    return h

# phantom gradient for the backward pass
# input: dl/ dh & output: dl / dz
def phantom_grad(g):
    # forward pass for automatic differentiation
    f = (1 - lambda_) * h + lambda_ * func(h, z)

    g_hat = g
    for _ in range(k-1):
        # compute Jacobian-vector product with automatic differentiation
        g_hat = g + grad(f, h, g_hat)

    # compute Jacobian-vector product to obtain dl / dz
    g_hat = grad(f, z, g_hat)
    return g_hat
```

Convergence Analysis

$$\frac{\widehat{\partial \mathcal{L}}}{\partial \theta} := \boldsymbol{A} \frac{\partial \mathcal{L}}{\partial h} \qquad \left\langle \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \frac{\widehat{\partial \mathcal{L}}}{\partial \mathbf{x}} \right\rangle > 0$$

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

$$\left\| \frac{\widehat{\partial \mathcal{L}}}{\partial \boldsymbol{\theta}} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} \right\| \le \varepsilon, \qquad \text{almost surely,}$$

- then using the phantom gradient as a stochastic first-order oracle with a step size of $\eta_\tau = O(1/\sqrt{\tau})$ to update $\boldsymbol{\theta}$ with gradient descent, it follows after T iterations that

$$\mathbb{E}\left[\frac{\sum_{\tau=1}^{T} \eta_\tau \|\nabla \mathcal{R}(\boldsymbol{\theta}_\tau)\|^2}{\sum_{\tau=1}^{T} \eta_\tau}\right] \le O\left(\varepsilon + \frac{\log T}{\sqrt{T}}\right).$$

Experiments

- Precision: the gap between the phantom gradient and the exact gradient?
  - Synthetic settings
  - Practical scenario

- Influences of hyperparameters k and λ?

- Computation cost
  - Phantom gradient compared with implicit differentiation?

- Phantom gradient at scale
  - Vision, Language, Graph
  - DEQ, MDEQ, IGNN
  - …

Static Precision

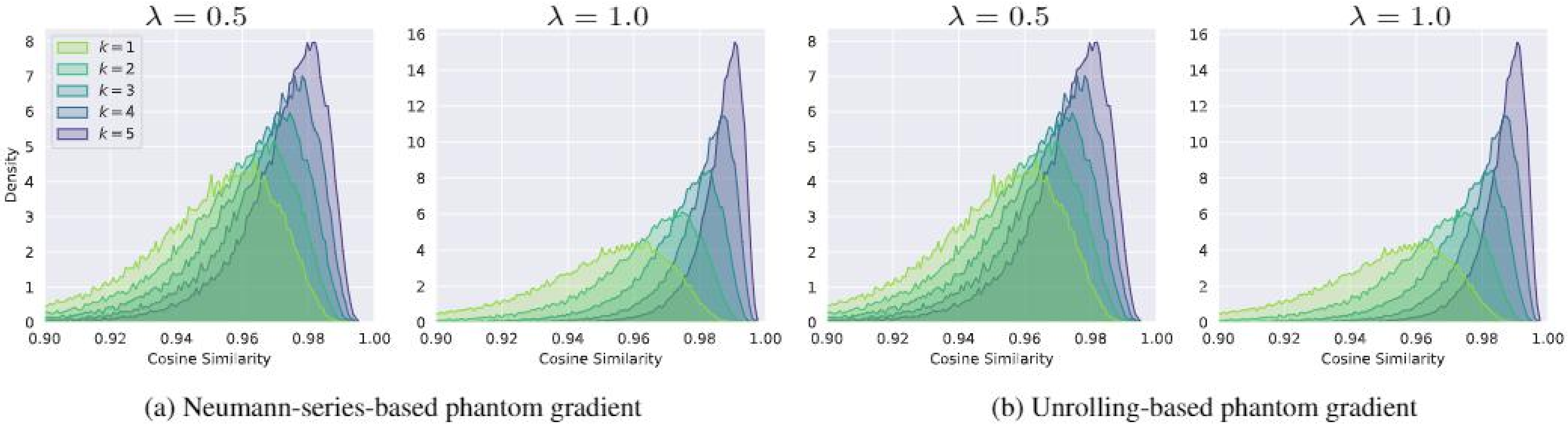
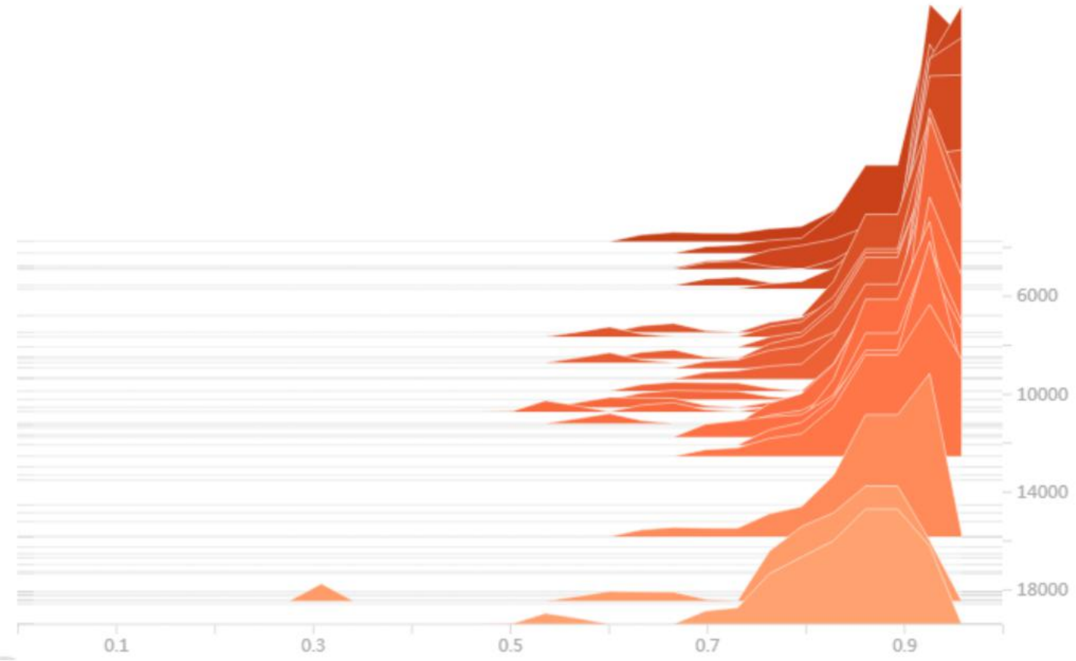


Figure 1: Cosine similarity between the phantom and the exact gradients in the synthetic setting.

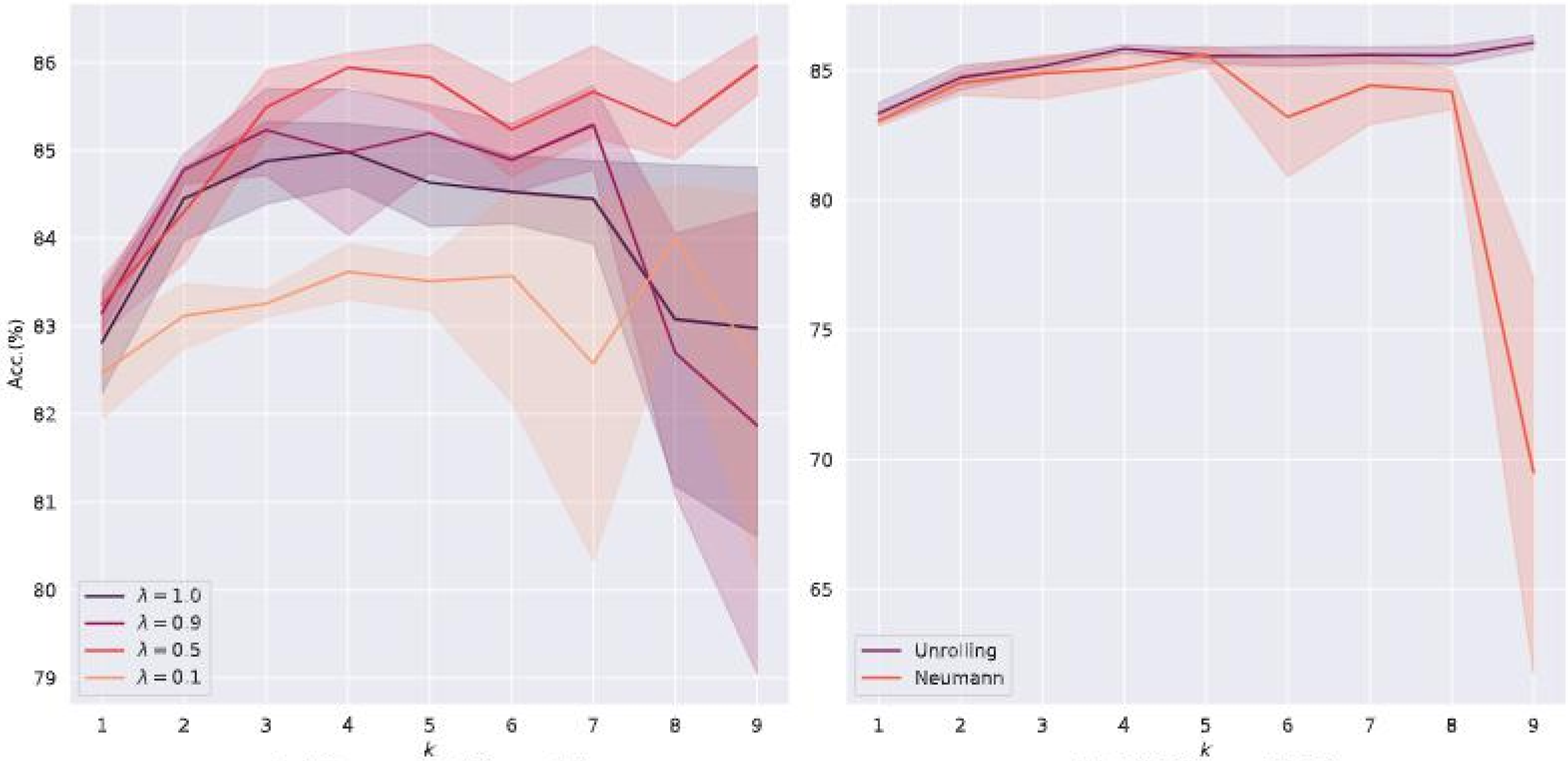
Dynamic Precision

- Histogram of cosine similarity between phantom gradient and implicit differentiation along the training.

- Phantom gradients preserve a high precision during the training dynamics.



Hyperparameters



(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.

Phantom Grad at Scale

Table 3: Large-scale experiments on CIFAR-10 and ImageNet classifications. Using phantom gradients, we are able to achieve comparable or better performance in these high-dimensional settings, while being much faster at training.

Task	**Method**	**Params**	**Acc(%)**	**Speed**	**Peak Mem**
CIFAR-10	MDEQ + Implicit	10M	93.8	1×	1×
CIFAR-10	MDEQ + UPG $\boldsymbol{A}_{5,0.5}$	10M	95.0	1.4×	0.5×
ImageNet	MDEQ + Implicit	18M	75.3	1×	1×
ImageNet	MDEQ + UPG $\boldsymbol{A}_{6,0.5}$	18M	75.7	1.7×	1×

12× acceleration
for the backward！

Phantom Grad at Scale

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 iterations in the forward equilibrium solver.

Datasets	Model	Method	Params	Metrics	Speed
CIFAR-10	MDEQ	Implicit	10M	93.8 ($\pm$0.17)	1×
CIFAR-10	MDEQ	UPG $\boldsymbol{A}_{5,0.5}$	10M	95.0 ($\pm$0.16)	1.4×
ImageNet	MDEQ	Implicit	18M	75.3	1×
ImageNet	MDEQ	UPG $\boldsymbol{A}_{5,0.6}$	18M	75.7	1.7×
Wikitext-103	DEQ (PostLN)	Implicit	98M	24.0	1×
Wikitext-103	DEQ (PostLN)	UPG $\boldsymbol{A}_{5,0.8}$	98M	25.7	1.7×
Wikitext-103	DEQ (PreLN)	JR + Implicit	98M	24.5	1.7×
Wikitext-103	DEQ (PreLN)	JR + UPG $\boldsymbol{A}_{5,0.8}$	98M	24.4	2.2×
Wikitext-103	DEQ (PreLN)	JR + UPG $\boldsymbol{A}_{5,0.8}$	98M	24.0†	1.7×

Phantom Grad at Scale

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 $\boldsymbol{A}_{5,1.0}$	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 $\boldsymbol{A}_{5,1.0}$	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 $\boldsymbol{A}_{5,1.0}$	4.7M	96.2

Take Away

- Precise gradient estimates are not always required, especially for a black box layer like our lovely implicit models!

- Phantom gradients can train implicit models to SOTA much faster.

- ...

- See more findings in our paper!

Thank you!