Meta-Learning with Adjoint Methods

Shibo Li[1], Zheng Wang[1], Akil Narayan[2,3], Robert M. Kirby[1,2] and Shandian Zhe[1]

[1]Kahlert School of Computing, [2]Scientific Computing and Imaging Institute, [3]Department of Mathematics;

University of Utah

{shibo, wzhut, kirby, zhe}@cs.utah.edu, akil@sci.utah.edu

Abstract

Model Agnostic Meta Learning (MAML) is widely used to find a good initialization for a family of tasks. Despite its success, a critical challenge in MAML is to calculate the gradient w.r.t. the initialization of a long training trajectory for the sampled tasks, because the computation graph can rapidly explode and the computational cost is very expensive. To address this problem, we propose Adjoint MAML (A-MAML). We view gradient descent in the inner optimization as the evolution of an Ordinary Differential Equation (ODE). To efficiently compute the gradient of the validation loss w.r.t. the initialization, we use the adjoint method to construct a companion, backward ODE. To obtain the gradient w.r.t. the initialization, we only need to run the standard ODE solver twice --- one is forward in time that evolves a long trajectory of gradient flow for the sampled task; the other is backward and solves the adjoint ODE. We need not create or expand any intermediate computational graphs, adopt aggressive approximations, or impose proximal regularizers in the training loss. Our approach is cheap, accurate, and adaptable to different trajectory lengths. We demonstrate the advantage of our approach in both synthetic and real-world meta-learning tasks. The code is available at https://github.com/shib0li/Adjoint-MAML.

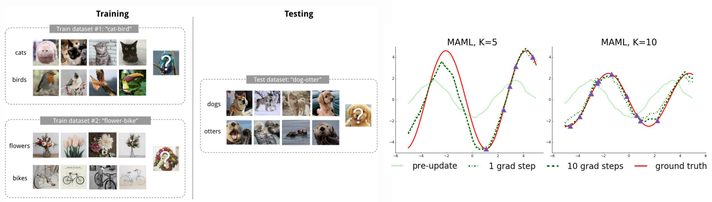
Motivation

Meta-Learning:

- Standard (Deep) Machine Learning: cheap, safe, easy to collect large amount of data

- Data Costly, Sensitive Applications: Robotics, User personalization, etc.
- ***Meta Learning(Learning to Learn Fast)***: Enable **efficient** learning on **new tasks** with encoding **adaptable representations**

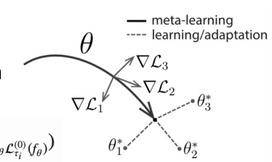
Examples: *Few-shot Classification; Few-shot Regression*



Meta-Learning Methods: *Metric* based; *Model* based; *Optimization* based

Mode-Agnostic Meta Learning

- A fairly *general* optimization algorithm
- Compatible with any model that learns through gradient descent



$$\theta^* = \arg\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)}(f_{\theta_i'}) = \arg\min_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}^{(0)}(f_\theta)})$$

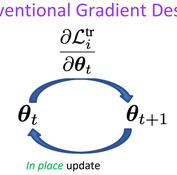
Task adaptation

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}^{(1)}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}^{(0)}(f_\theta)})$$

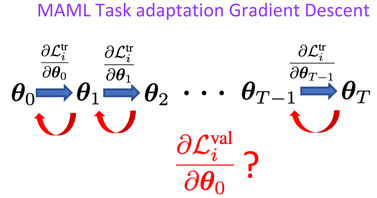
Meta update/optimization

- Exhaustive memory usage due to saving the entire gradient descent trajectory.

Conventional Gradient Descent

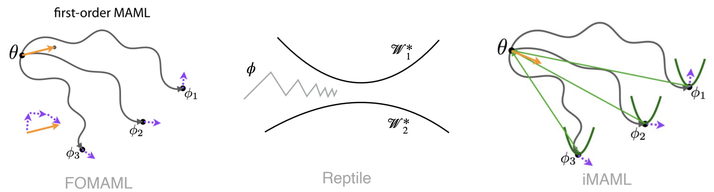


MAML Task adaptation Gradient Descent



Approximations of MAML

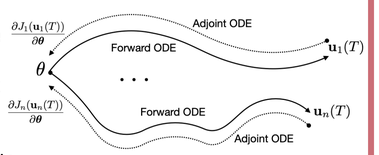
- First-order MAML; Reptile; Meta-Learning with Implicit Gradient (iMAML)



- One or few inner steps is/are sufficient to approximate the meta gradients. (original MAML)
- Don't look back at all. Ignores the outer level optimization. (FOMAML, Reptile, iMAML)
- Strong (local) curvature heuristic about the meta gradients (iMAML)

Our Contribution: Adjoint-MAML

- Forward ODE: Task adaptation (gradient flow)
- (Backward) Adjoint ODE: Dynamics of the meta-gradient
- Resurgence the "ideal" MAML (trace back the exact task adaptation)
- Memory efficiency for long adaptation trajectory while yet accurate meta gradients on validation



Methods

Adjoint Method:

$$\begin{cases} \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{f}(\mathbf{u}, t; \boldsymbol{\theta}) \\ \mathbf{u}(t_0) = \mathbf{u}_0 \end{cases} \quad \mathbf{u}(t) \in \mathbb{R}^N; \mathbf{f} \in \mathbb{R}^N; \boldsymbol{\theta} \in \mathbb{R}^P$$

$$\min_{\boldsymbol{\theta}} J(\mathbf{u}, \boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \int_0^T g(\mathbf{u}, \boldsymbol{\theta}) \mathrm{d}t$$

For example: Quadratic Loss: $\mathbf{u}^\top \mathbf{Q} \mathbf{u}$

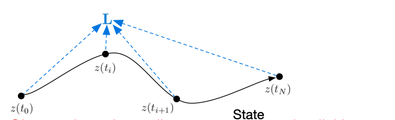
Goal: To Compute the total derivative

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\boldsymbol{\theta}} = [\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\theta_0}, \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\theta_1}, \cdots, \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\theta_P}]$$

$$\frac{\mathrm{d}}{\mathrm{d}\theta_i} \begin{cases} \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{f}(\mathbf{u}, t; \boldsymbol{\theta}) \\ \mathbf{u}(t_0) = \mathbf{u}_0 \end{cases}$$

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t} \cdot \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\theta_i} = \frac{\mathrm{df}}{\mathrm{d}\mathbf{u}} \cdot \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}\theta_i} + \frac{\mathrm{df}}{\mathrm{d}\theta_i} \\ \frac{\mathrm{d}\mathbf{u}(0)}{\mathrm{d}\theta_i} = \frac{\mathrm{d}\mathbf{u}_0}{\mathrm{d}\theta_i} \end{cases} \quad \mathbf{s}_i$$

Solve P+1 ODE systems



Observations depending on parameters implicitly

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta) dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{\theta}} = \int_0^T (\frac{\partial g}{\partial \boldsymbol{\theta}} + \frac{\partial g}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial \boldsymbol{\theta}}) \mathrm{d}t$$

(1xP, 1xP, 1xN, NxP)

$$\hat{J}(\mathbf{u}; \boldsymbol{\theta}) = J(\mathbf{u}; \boldsymbol{\theta}) + \int_0^T \boldsymbol{\lambda}^\top(t)(\mathbf{f} - \frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t}) \mathrm{d}t$$

Auto-differentiation

T.V.P

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{\lambda}(t)}{\mathrm{d}t} = -\frac{\mathrm{df}}{\mathrm{du}}^\top \cdot \boldsymbol{\lambda}(t) - \frac{\mathrm{dg}}{\mathrm{du}} \\ \boldsymbol{\lambda}(T) = \mathbf{0} \end{cases}$$

$$\frac{\mathrm{d}\hat{J}}{\mathrm{d}\boldsymbol{\theta}} = \int_0^T (\frac{\mathrm{dg}}{\mathrm{d}\boldsymbol{\theta}} + \boldsymbol{\lambda}(t) \frac{\mathrm{df}}{\mathrm{d}\boldsymbol{\theta}}) \mathrm{d}t + \boldsymbol{\lambda}(0) \frac{\mathrm{d}\mathbf{u}_0}{\mathrm{d}\boldsymbol{\theta}}$$

Solve **only 2** ODE systems, scale constantly

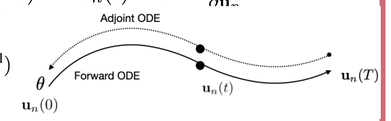
Adjoint Model Agnostic Meta-Learning (AMAML):

Step 1: Task adaptation, Forward ODE (Gradient Flow)

I.V.P

$$\begin{cases} \mathbf{u}_n(0) &= \boldsymbol{\theta}, \\ \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t} &= -\frac{\partial \mathcal{L}(\mathbf{u}_n, \mathcal{D}_n^{\text{tr}})}{\partial \mathbf{u}} \end{cases} \quad \mathbf{u}_n(t+\alpha) \leftarrow \mathbf{u}_n(t) - \alpha \frac{\partial \mathcal{L}(\mathbf{u}_n, \mathcal{D}_n^{tr})}{\partial \mathbf{u}_n}$$

Validation loss $J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\mathbf{u}_n(T), \mathcal{D}_n^{\text{val}})$



Step 2: Derived the Adjoint ODE

Target Constrained Optimization problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}[J(\boldsymbol{\theta})] = \min_{\boldsymbol{\theta}} \frac{1}{n} \sum_n \mathcal{L}(\mathbf{u}_n(T), \mathcal{D}^{\text{val}})$$

$$\text{s.t. } \forall \mathcal{T}_n \sim p(\mathcal{T})$$

$$\mathbf{u}(0) = \boldsymbol{\theta}$$

$$\frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t} = \mathbf{f}(\mathbf{u}_n, \mathcal{D}_n^{\text{tr}})$$

Lagrangian Relaxation

$$\hat{J}_n = J_n(\mathbf{u}_n(T)) + \int_0^T \boldsymbol{\lambda}(t)^\top \left(f(\mathbf{u}_n, \mathcal{D}_n^{\text{tr}}) - \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t}\right) \mathrm{d}t.$$

Sensitivity Cancellation

$$\frac{\mathrm{d}J_n}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial J_n}{\partial \mathbf{u}_n(T)} \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}\boldsymbol{\theta}}(T) - \boldsymbol{\lambda}(T)^\top \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}\boldsymbol{\theta}}(T) + \boldsymbol{\lambda}(0)^\top \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}\boldsymbol{\theta}}(0) + \int_0^T \left\{ \boldsymbol{\lambda}^\top \frac{\partial \mathbf{f}}{\mathbf{u}_n} \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}\boldsymbol{\theta}} + \left(\frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t}\right)^\top \frac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}\boldsymbol{\theta}} \right\} \mathrm{d}t$$

T.V.P

$$\begin{cases} \boldsymbol{\lambda}(T) &= \left(\frac{\partial J_n}{\partial \mathbf{u}_n(T)}\right)^\top \\ \left(\frac{\mathrm{d}\boldsymbol{\lambda}}{\mathrm{d}t}\right)^\top &= -\boldsymbol{\lambda}(t)^\top \frac{\partial \mathbf{f}}{\partial \mathbf{u}_n}, \\ \frac{\partial \mathbf{f}}{\partial \mathbf{u}_n} = \mathbf{H}(\mathbf{u}_n) = -\frac{\partial^2 \mathcal{L}(\mathbf{u}_n, \mathcal{D}_i^{tr})}{\partial \mathbf{u}_n^2} \end{cases}$$

Experiment

Evaluation 1: **Regression Tasks**

(a) *CosMixture* instance (b) *CosMixture*: 50shot-50val (d) *Alpine* instance (e) *Alpine*: 50shot-50val

Evaluation 2: **Collaborative Filtering**

	Jester-1		MovieLens100K		MovieLens1M	
	10shot-15val	20shot-30val	10shot-15val	20shot-30val	10shot-15val	20shot-30val
A-MAML	**0.074±0.005**	**0.027±0.002**	**0.053±0.005**	**0.023±0.003**	**0.094±0.008**	**0.035±0.004**
iMAML	0.114±0.007	0.050±0.003	0.082±0.004	0.033±0.002	0.138±0.010	0.052±0.004
MAML	0.120±0.001	0.036±0.000	0.123±0.001	0.050±0.003	0.140±0.002	0.059±0.001
FOMAML	0.292±0.012	0.115±0.004	0.174±0.008	0.068±0.004	0.270±0.011	0.104±0.006
Reptile	0.270±0.012	0.106±0.004	0.166±0.008	0.063±0.003	0.266±0.011	0.101±0.006

Table 1: Meta-test error (nRMSE) with 50 inner gradient descent steps (MAML used 5 GD steps) The results were averaged over 100 tasks.

Evaluation 3: **Image Classification**

Method	Ominiglot	Mini-ImageNet
MAML	$95.8 \pm 0.3\%$	$48.70 \pm 1.84\%$
FOMAML	$89.4 \pm 0.5\%$	$48.07 \pm 1.75\%$
Reptile	$89.43 \pm 0.14\%$	$\mathbf{49.97 \pm 0.32\%}$
iMAML-GD	$94.46 \pm 0.42\%$	$48.96 \pm 1.84\%$
iMAML-HF	$96.18 \pm 0.36\%$	$49.30 \pm 1.88\%$
A-MAML($T = 0.5$)	$96.36 \pm 0.39\%$	$49.43 \pm 1.64\%$
A-MAML($T = 1.0$)	$\mathbf{96.79 \pm 0.34\%}$	$49.47 \pm 1.77\%$

Table 3: Meta-test accuracy for 20-way 1-shot on *Omniglot* and 5-way 1-shot on *Mini-ImageNet*.

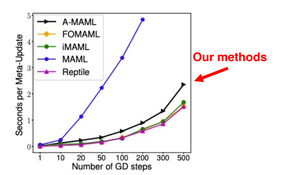


Figure 4: Running time of the inner gradient descent for *CosMixture*.

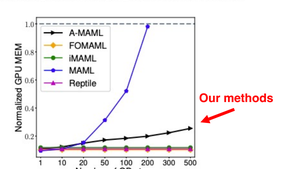


Figure 3: Normalized GPU usage in meta learning of *CosMixutre* with 100shot-100validation. The dashed line indicates the capacity of available GPU memory.

Evaluation 4: **Efficiency Analysis**

Evaluation 5: **Trade-off of the Meta-Gradients**

(Appendix A of the paper)