Meta-Learning with Adjoint Method

Shibo Li, Zheng Wang, Akil Narayan, Robert M. Kirby, Shandian Zhe

The University of Utah

Roadmap

- Motivation
- Background
- Method
- Experiment
- Light Discussion

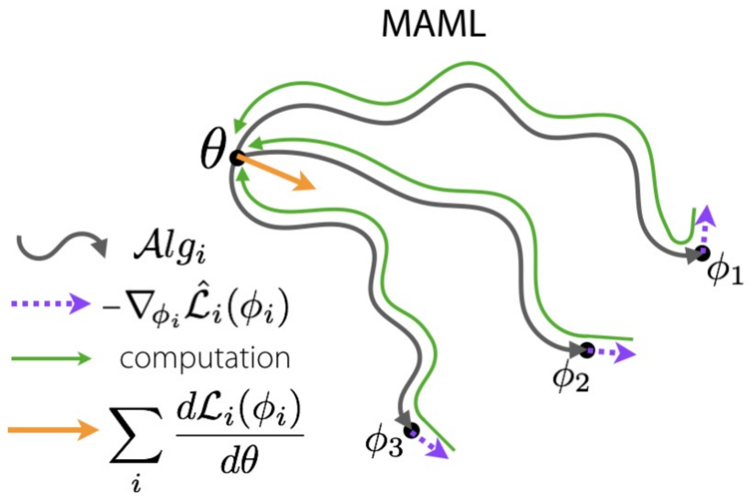
Motivation

- Meta-Agnostic Meta Learning(MAML)
  - Task adaptation:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

  - Meta-optimization:

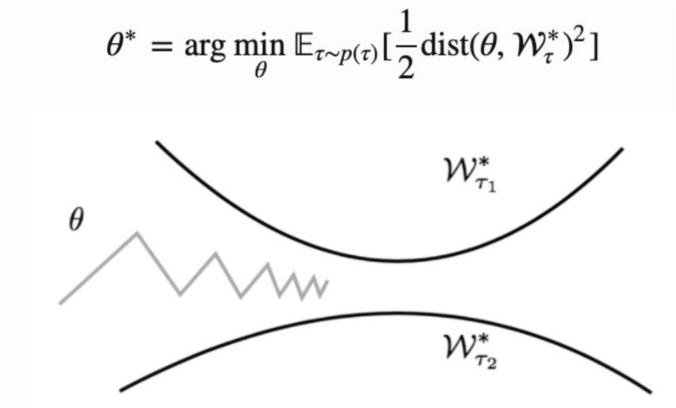
$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$



Rajeswaran et al. 2019

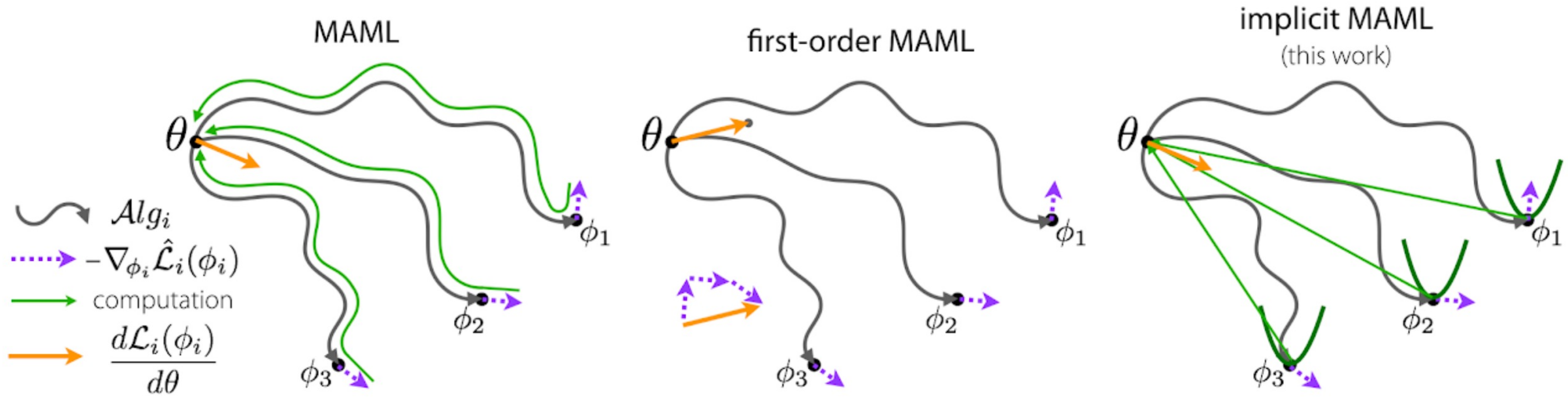
  - The backward requires taking auto differentiation on the history of computational graph.

Motivation



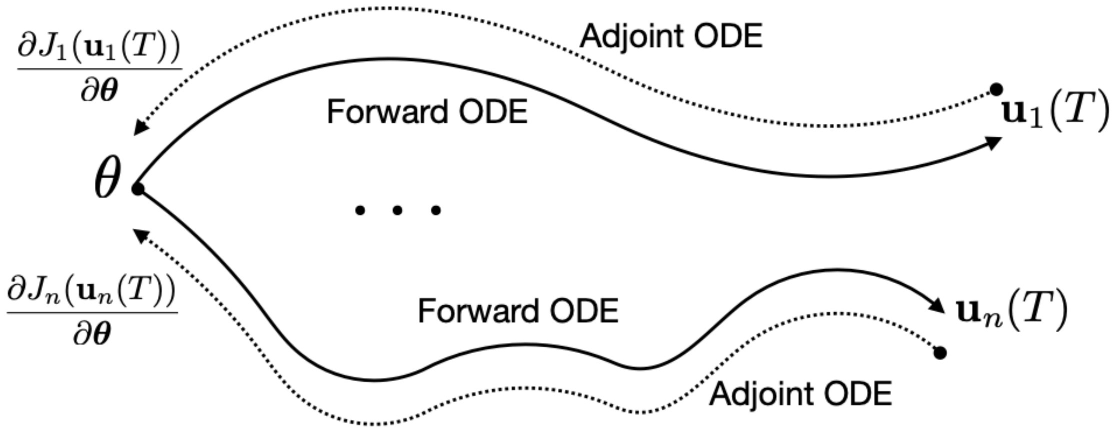
Reptile, Nichol et al. 2018

- Existing approximation:
  - First-order MAML: no backward
  - Implicit MAML: local curvature $\frac{\lambda}{2}||\boldsymbol{\phi}' - \boldsymbol{\theta}||^2$
  - Reptile: close to all the optimal manifolds of all tasks



Our Contribution

- An ODE view of task adaptation
- Meta gradient with adjoint method
- Memory efficiency for long adaptation trajectory while yet accurate meta gradients on validation



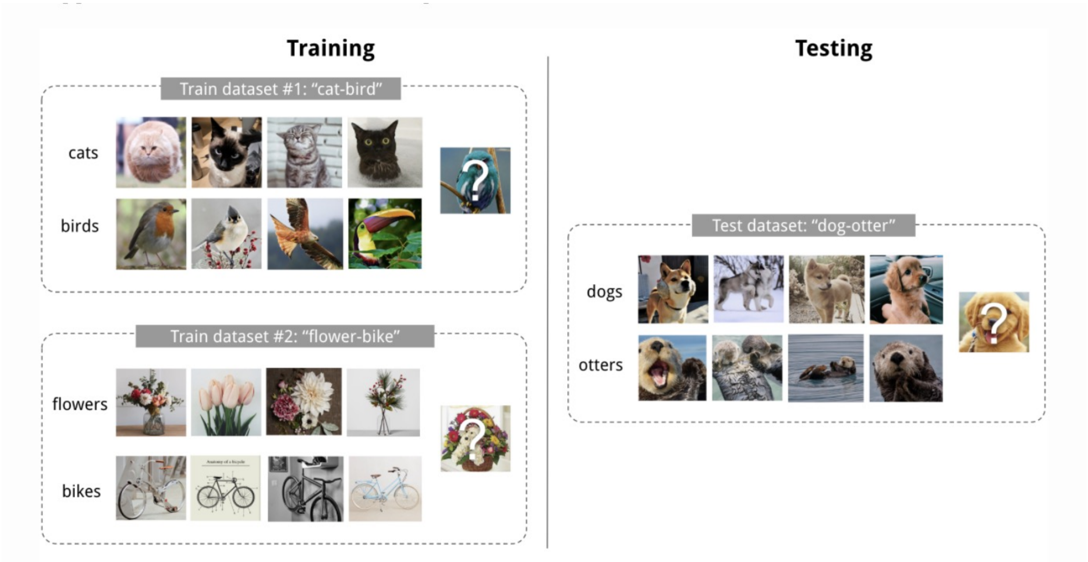
Background

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

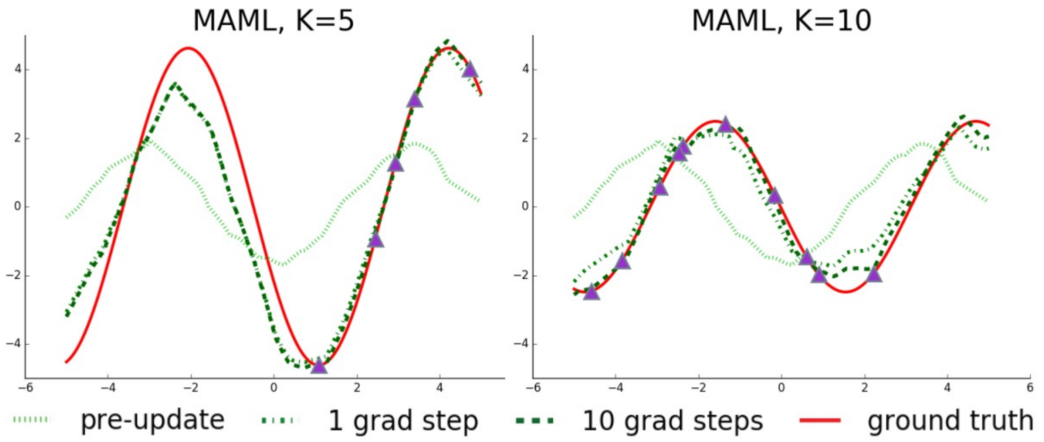
- Data Costly, Sensitive Applications: Robotics, User personalization, etc.
- ***Meta Learning(Learning to Learn Fast)***: Enable **efficient** learning on **new tasks** with encoding **adaptable representations**

Background

- Few-shot Classification (2way-4shots)

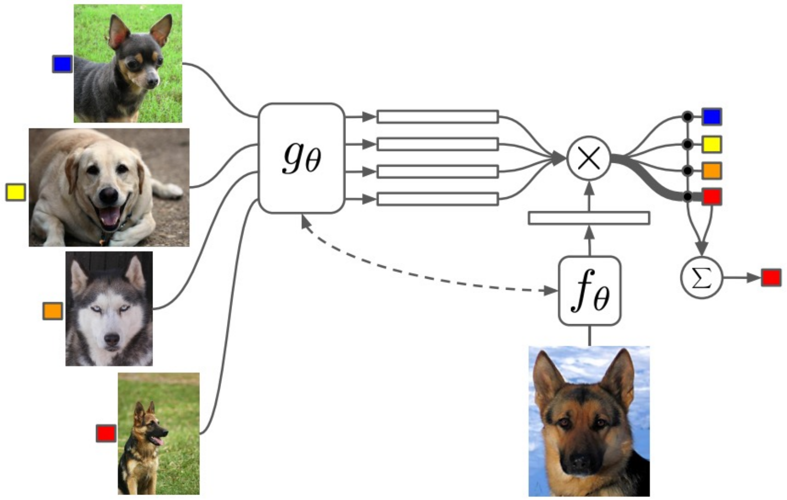


- Few-shot Regression

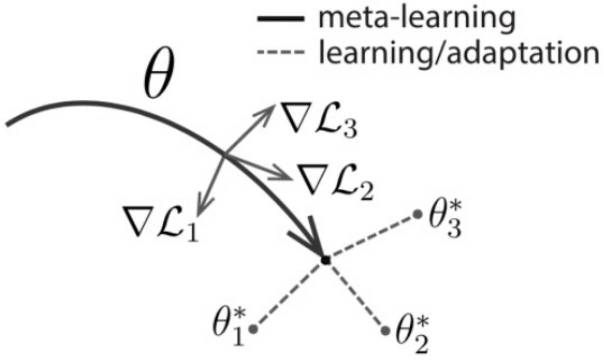


Meta-Learning Approaches

- *Metric Based*: Siamese Neural Network, Matching Net, etc.
- *Optimization Based*: **MAML**, Reptile, etc.



Matching Net



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

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

Model Agnostic Meta-Learning (MAML)

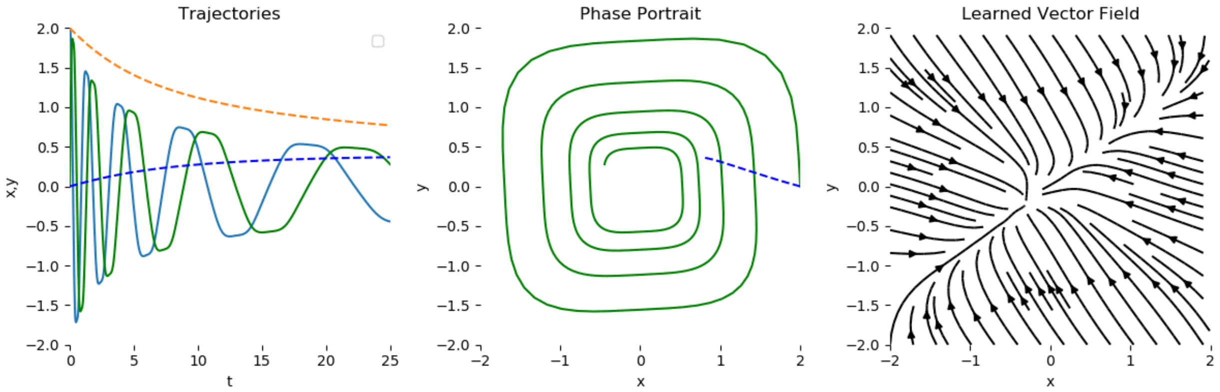
Adjoint Method

- An Example: Learn a parametric ODE system

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

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

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



Chen et al. 2018

Adjoint Method

- Goal: Total derivative $$\underbrace{\frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{\theta}}}_{1\mathrm{xP}} = \int_0^T (\underbrace{\frac{\partial g}{\partial \boldsymbol{\theta}}}_{1\mathrm{xP}} + \underbrace{\frac{\partial g}{\partial \mathbf{u}}}_{1\mathrm{xN}} \cdot \underbrace{\frac{\partial \mathbf{u}}{\partial \boldsymbol{\theta}}}_{\mathrm{NxP}})\mathrm{d}t$$

Forward Sensitivity

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

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

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

Solve P+1 ODE systems

Adjoint Sensitivity

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

Auto-differentiation

T.V.P $$\begin{cases} \frac{\mathrm{d}\boldsymbol{\lambda}(t)}{\mathrm{d}t} = -\frac{\mathrm{d}\mathbf{f}}{\mathrm{d}\mathbf{u}}^\top \cdot \boldsymbol{\lambda}(t) - \frac{\mathrm{d}g}{\mathrm{d}\mathbf{u}} \\ \boldsymbol{\lambda}(T) = \mathbf{0} \end{cases}$$

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

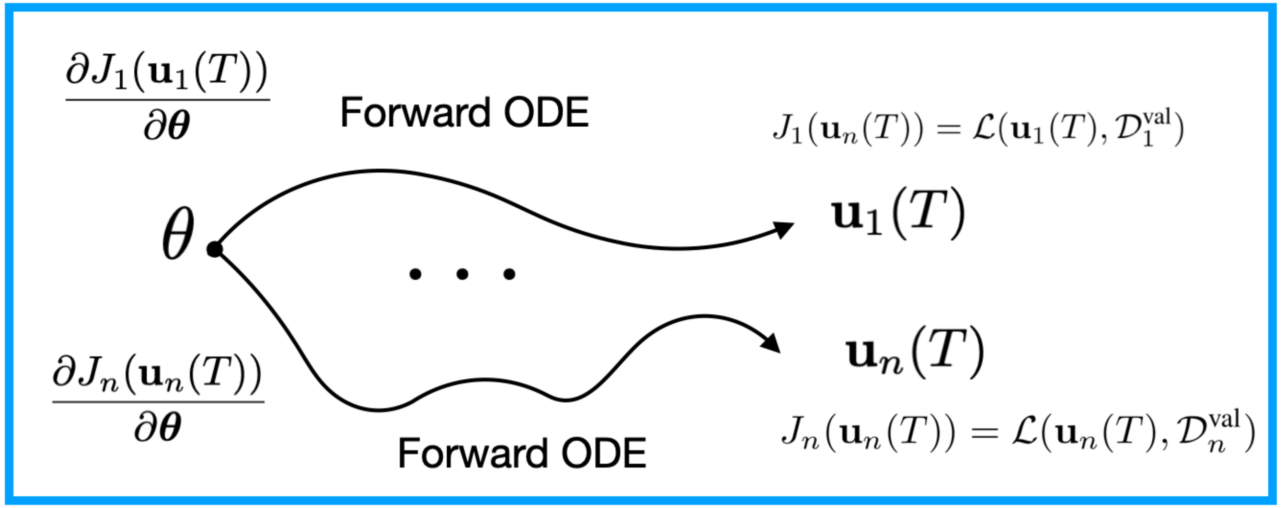
Solve **only 2** ODE systems, scale constantly 🙂

An ODE View of Task Adaptation

- Forward Propagation

I.V.P
$$\begin{cases} \mathbf{u}_n(0) &= \boldsymbol{\theta}, \\ \dfrac{\mathrm{d}\mathbf{u}_n}{\mathrm{d}t} &= -\dfrac{\partial \mathcal{L}(\mathbf{u}_n, \mathcal{D}_n^{\text{tr}})}{\partial \mathbf{u}} \end{cases}$$

$$\mathbf{u}_n(t+\alpha) \leftarrow \mathbf{u}_n(t) - \alpha \frac{\partial \mathcal{L}(\mathbf{u}_n, \mathcal{D}_n^{tr})}{\partial \mathbf{u}_n}$$



- Validation-Loss

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

Meta-Loss Minimization

- Target Problem

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

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

1xd vector

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

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

- Lagrangian relaxation

How to optimize? GD! n.t.s $\frac{\mathrm{d}\hat{J}_n}{\mathrm{d}\boldsymbol{\theta}}$

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

Meta-Loss Minimization

- Sensitivity/Jacobian Cancellation

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

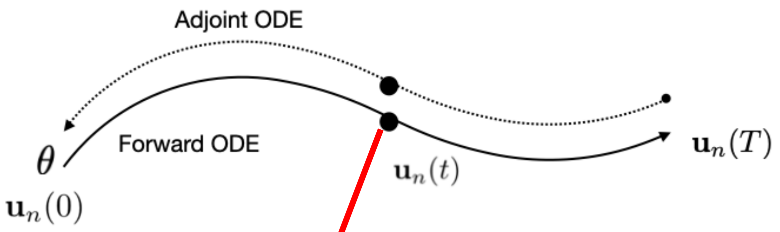
- Adjoint ODE

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

Meta-Loss Minimization

- Efficient Backpropagation

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

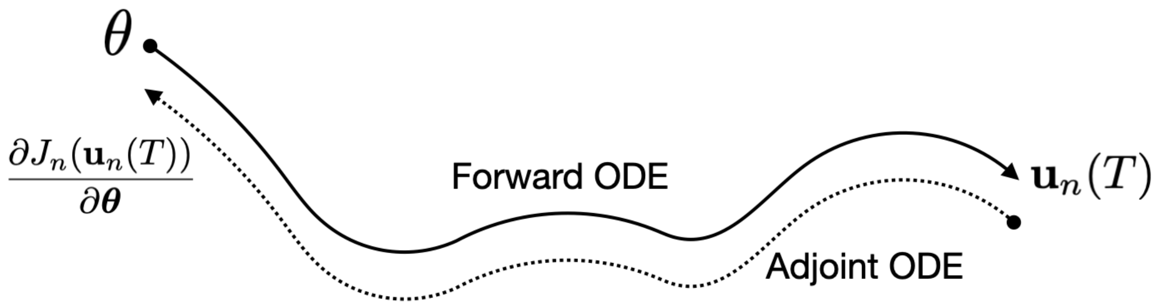


$$\widetilde{\boldsymbol{\lambda}}_j = \boldsymbol{\lambda}_{j+1} + h\mathbf{H}(\mathbf{u}_{n,j+1})\boldsymbol{\lambda}_{j+1},$$

$$\boldsymbol{\lambda}_j = \boldsymbol{\lambda}_{j+1} + \frac{h}{2}\left[\mathbf{H}(\mathbf{u}_{n,j+1})\boldsymbol{\lambda}_{j+1} + \mathbf{H}(\mathbf{u}_{n,j})\widetilde{\boldsymbol{\lambda}}_j\right]$$

Quick Summary

- “Continuous” task adaptation
- Constrained meta-loss minimization with J_n
- Relaxed meta-loss minimization with $\widehat{J}_n$
- GD on meta-loss requires $\frac{\mathrm{d}\hat{J}_n}{\mathrm{d}\boldsymbol{\theta}}$
- Two ODE systems
  - **I.V.P**: Forward, task adaptation
  - **T.V.P**: Backward, gradient of meta-loss

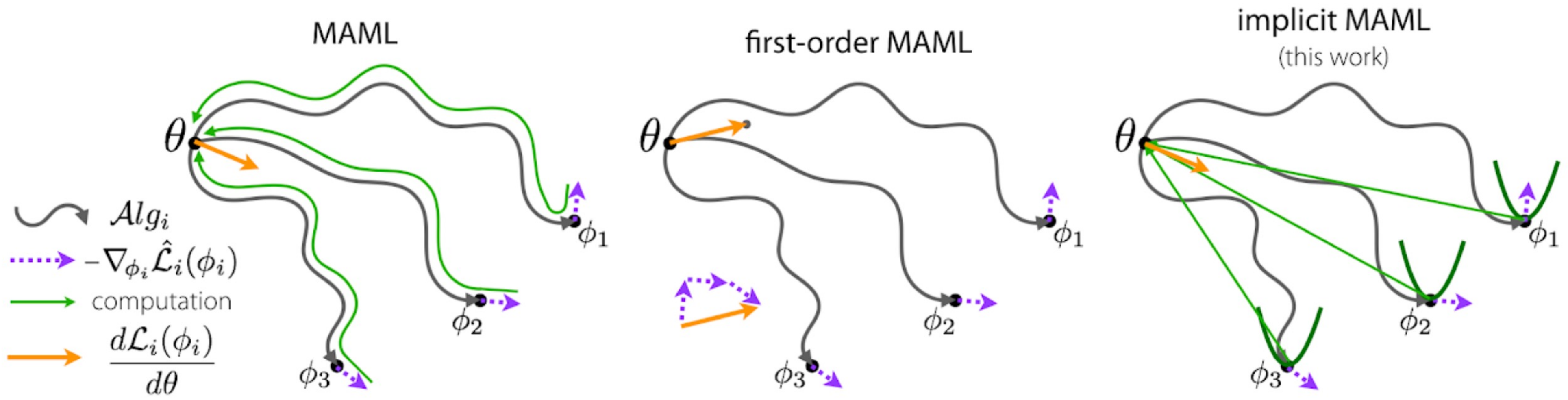


Experiment

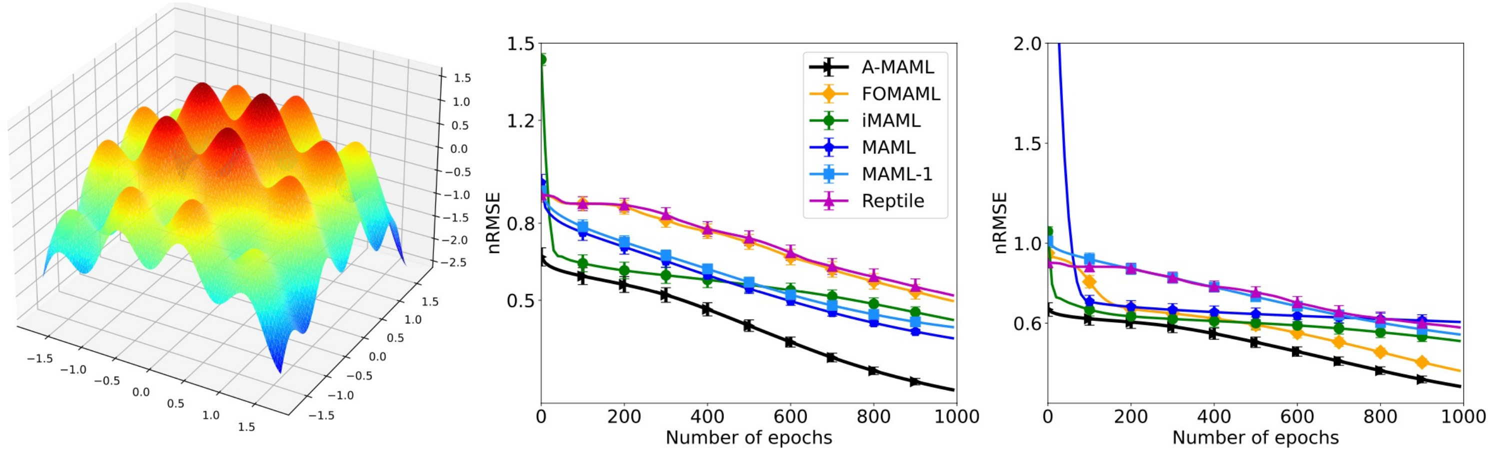
- How does our Adjoint-MAML (A-MAML) perform on synthetic problems?
- Efficiency of long adaptation trajectory:
  - Memory
  - Time
- How does A-MAML perform on real-world problems?

Experiment

- Comparing Methods
  - MAML, MAML(1GD)
  - First Order MAML
  - Implicit MAML
  - Reptile



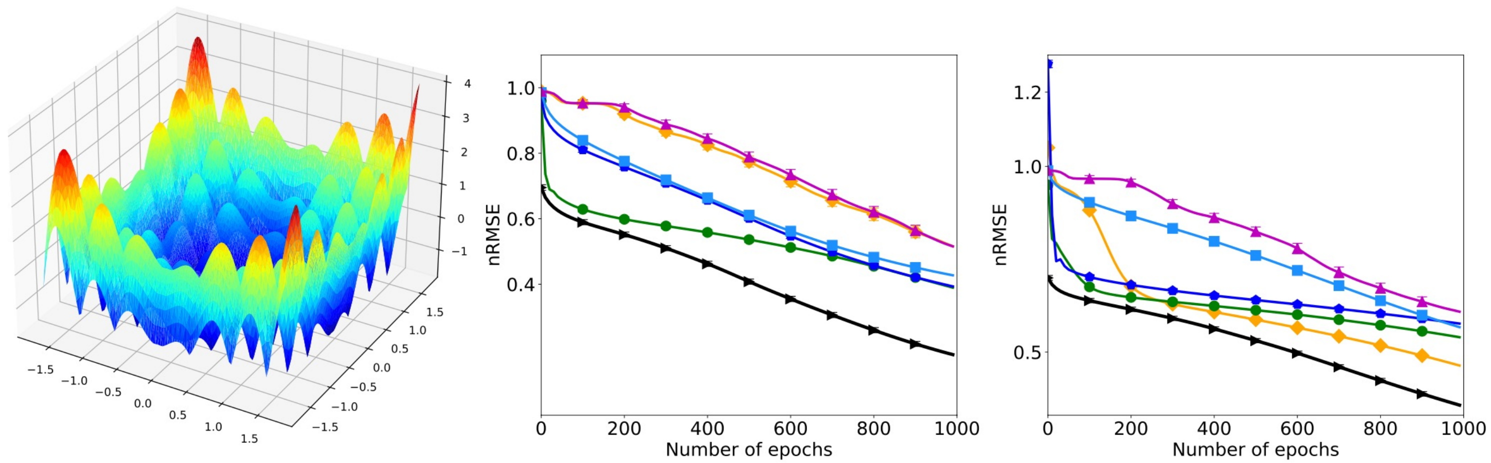
Synthetic Meta Regression



(a) *CosMixture* instance (b) *CosMixture*: 50shot-50val (c) *CosMixture*: 100shot-100val

$$f_1(\mathbf{x}) = -0.1 \sum_{i=1}^{d} A \cos(\omega x_i + \phi) - \sum_{i=1}^{d} x_i^2$$

$$A \in [0.1, 1.0]$$
$$\omega \in [0.5\pi, 2.0\pi]$$
$$\phi \in [3.0, 6.0]$$



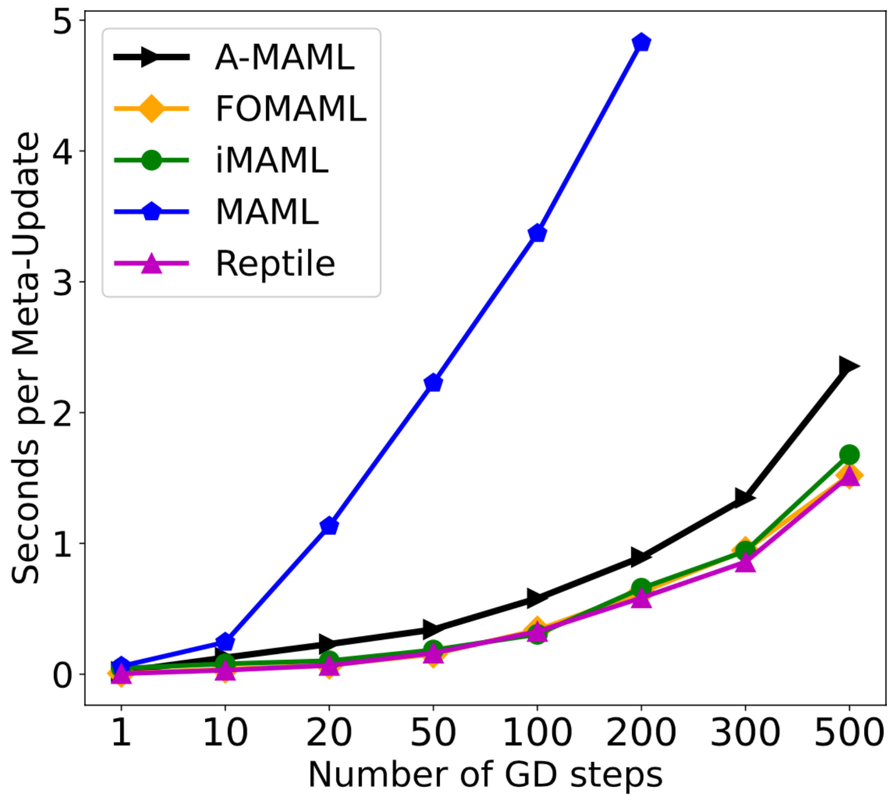
(d) *Alpine* instance (e) *Alpine*: 50shot-50val (f) *Alpine*: 100shot-100val

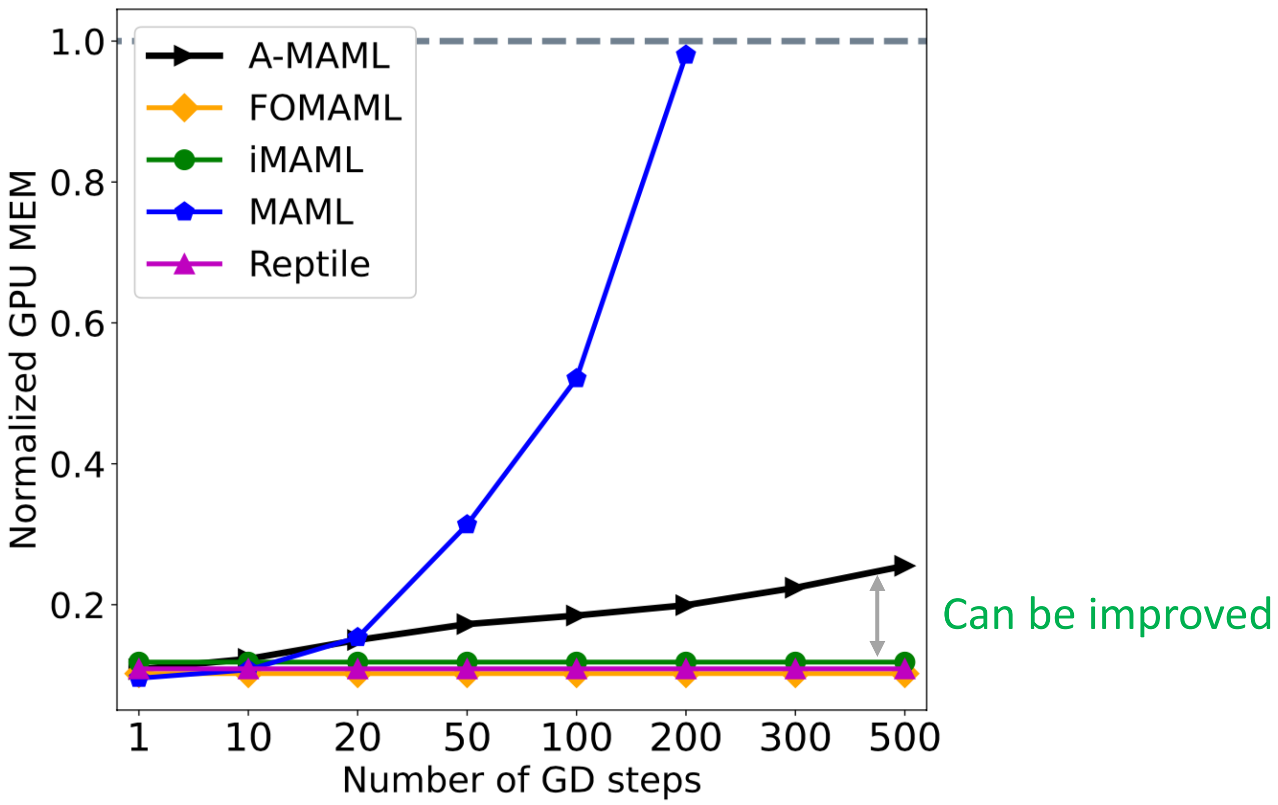
$$f_2(\mathbf{x}) = \sum_{i=1}^{d} |x_i \sin(x_i + \phi_i) + 0.1 x_i|$$

$$\phi_1 \in [-\frac{5}{12}\pi, \frac{5}{12}\pi]$$
$$\phi_2 \in [-\frac{5}{12}\pi, \frac{5}{12}\pi]$$

Synthetic Meta Regression

- Time/Memory Efficiency





Meta Collaborative Filtering

- Dataset: Jester-1, MovieLens100K, MovieLens1M
- Each user defines a task

	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.

Meta Collaborative Filtering

	Jester-1		MovieLens100K		MovieLens1M	
	10shot-15val	20shot-30val	10shot-15val	20shot-30val	10shot-15val	20shot-30val
A-MAML	**0.069±0.005**	**0.044±0.003**	**0.057±0.006**	**0.021±0.002**	**0.105±0.009**	**0.035±0.004**
iMAML	0.190±0.010	0.103±0.005	0.168±0.007	0.046±0.002	0.130±0.007	0.045±0.004
MAML	0.154±0.001	0.061±0.002	0.123±0.001	0.050±0.002	0.197±0.002	0.083±0.001
FOMAML	0.273±0.012	0.077±0.004	0.191±0.007	0.071±0.004	0.395±0.010	0.119±0.005
Reptile	0.290±0.012	0.100±0.004	0.171±0.008	0.066±0.004	0.408±0.011	0.128±0.006

Table 2: Meta-test error (nRMSE) with 100 inner gradient descent steps (MAML used 10 GD steps) The results were averaged over 100 tasks.

Conclusion

- A-AMAL accurately estimates the gradient of meta-loss
- Time and memory efficiency are compatible with other raw approximations
- Dominant performance on synthetic and real-world applications