

# **MAML: Model-Agnostic Meta-Learning for Fast Adaptation for Deep Networks**

HyeongJoo Hwang  
Jeonggwan Lee  
Joonyoung Yi

2018. 11. 28.

# Contents

1. Meta-Learning Problem Setup (Definition & Goal)
2. Prior Work
3. Model Agnostic Meta Learning
  1. Characteristics
  2. Intuition
4. Approach
  1. Supervised Learning
  2. Gradient of Gradient
  3. Reinforcement Learning
5. Experiment
6. Discussion

## 1. Problem Set-up

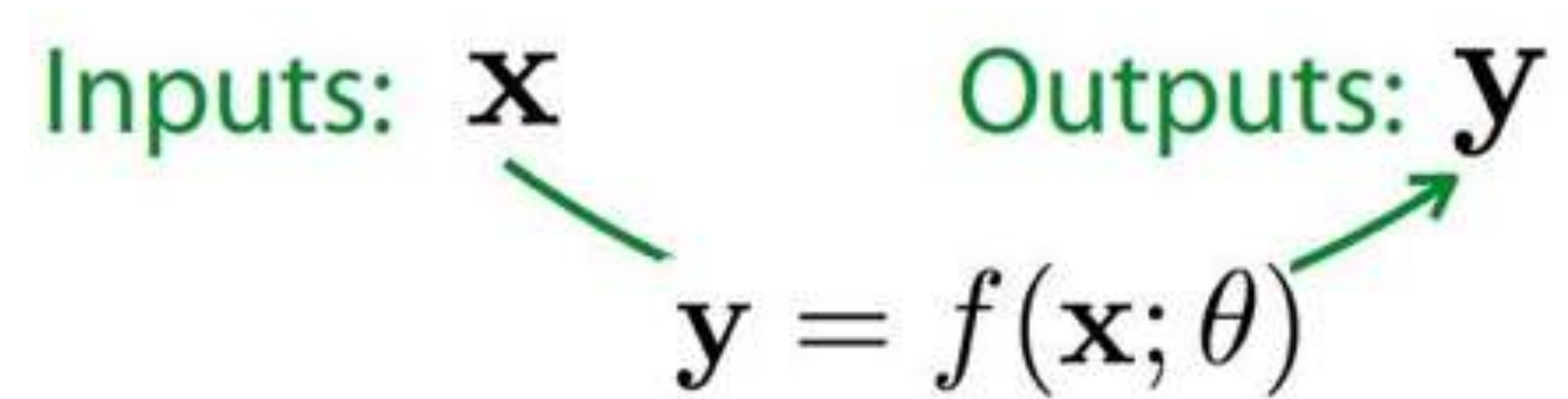
# Meta-Learning Problem Set-up

- Definition of meta-learning:
  - Learner is trained by meta-learner to be able to learn on many different tasks.
- Goal:
  - Learner quickly learn new tasks from a small amount of new data.

## 1. Problem Set-up

# Meta Supervised Learning

- Supervised learning:

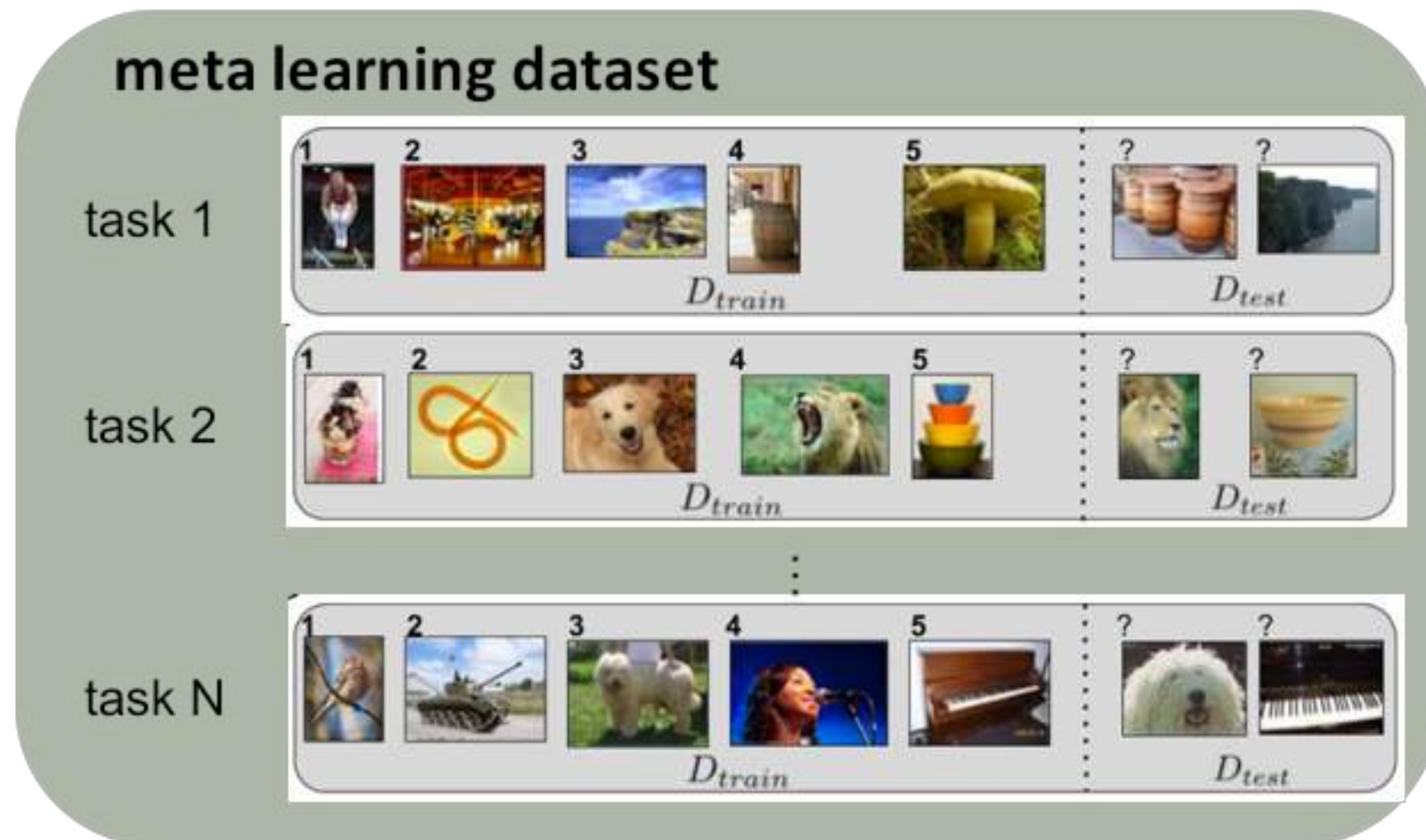


Data:  $\{(\mathbf{x}, \mathbf{y})_i\}$

- Meta supervised learning?

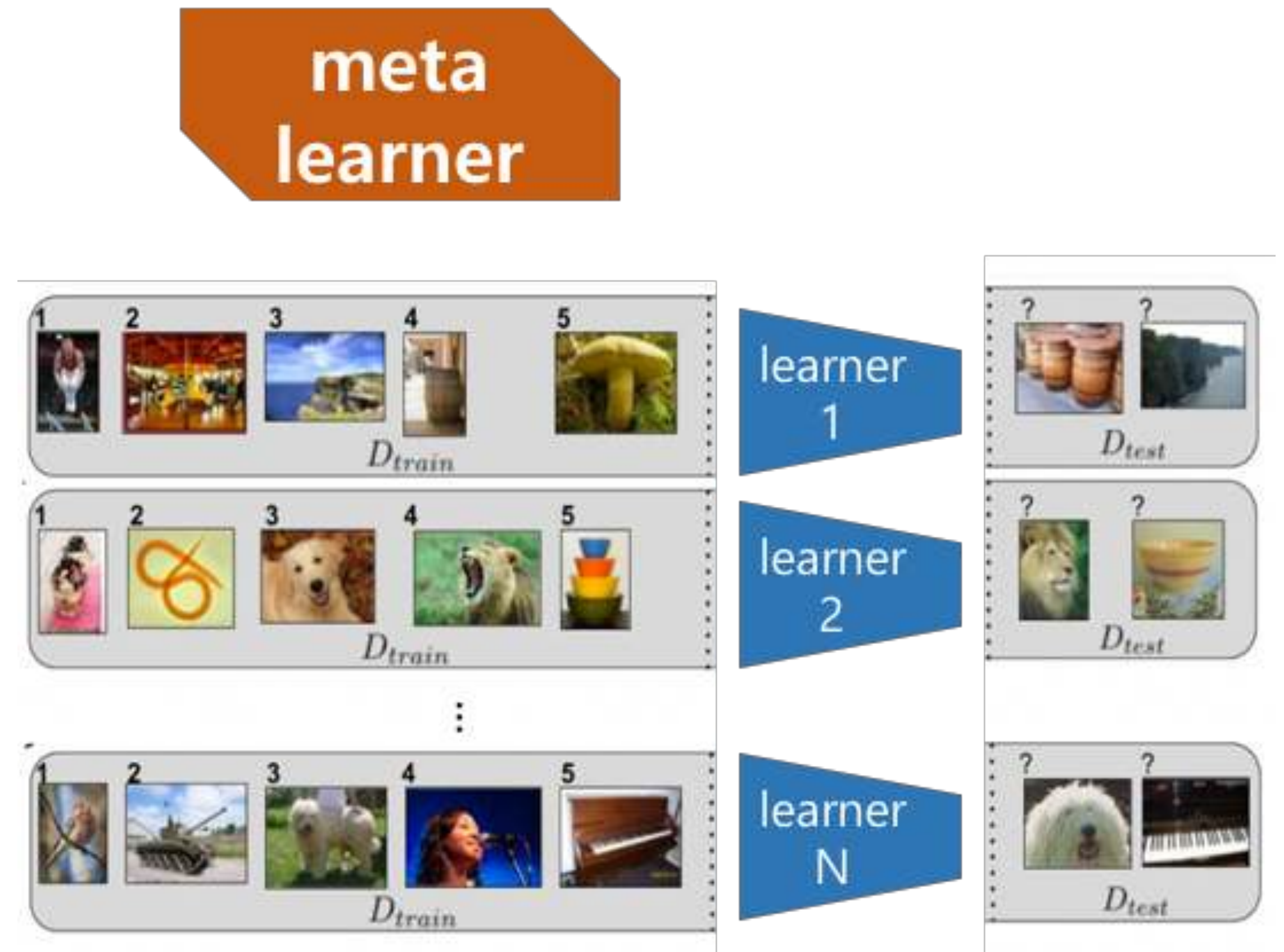
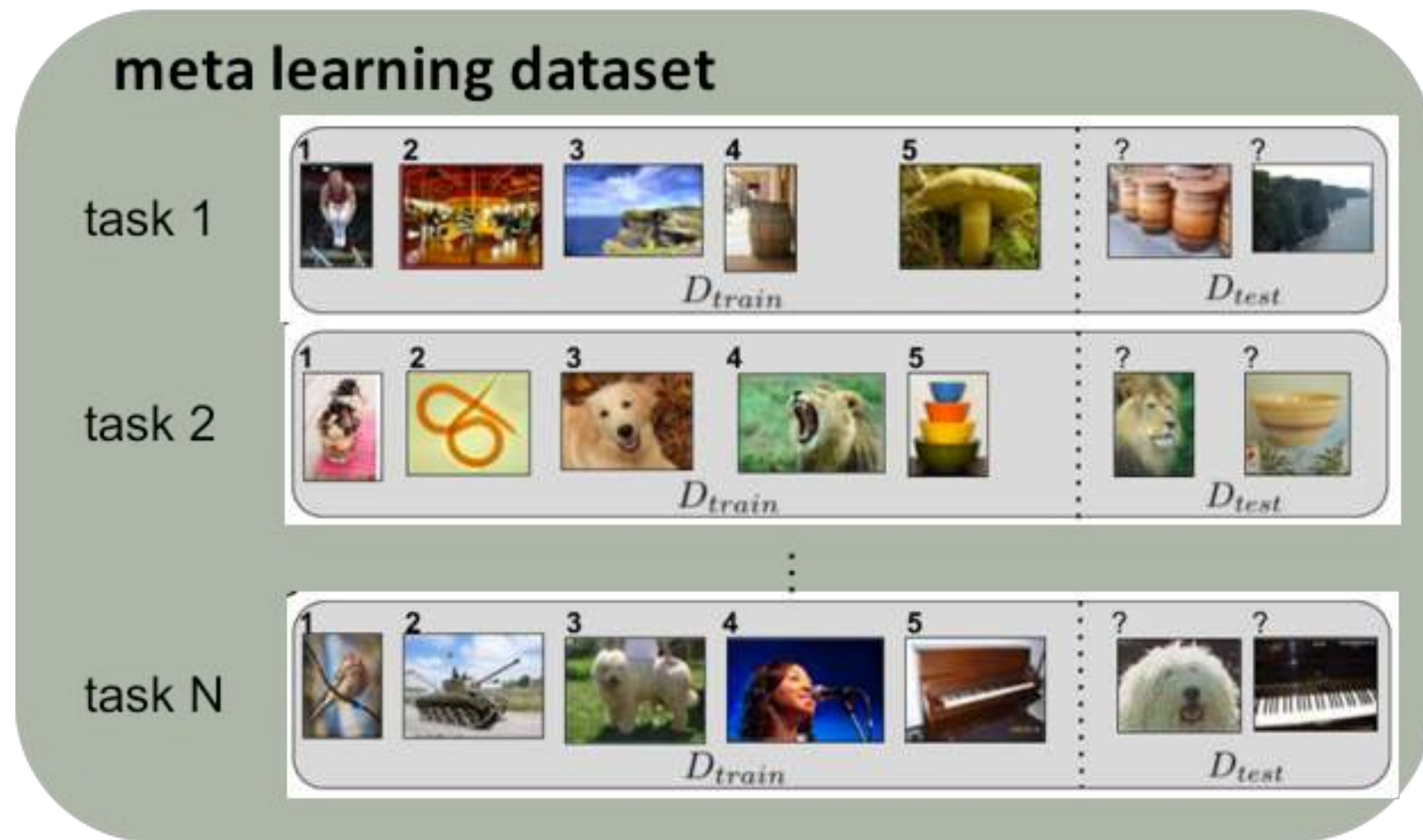
1. Problem Set-up

# Meta Supervised Learning



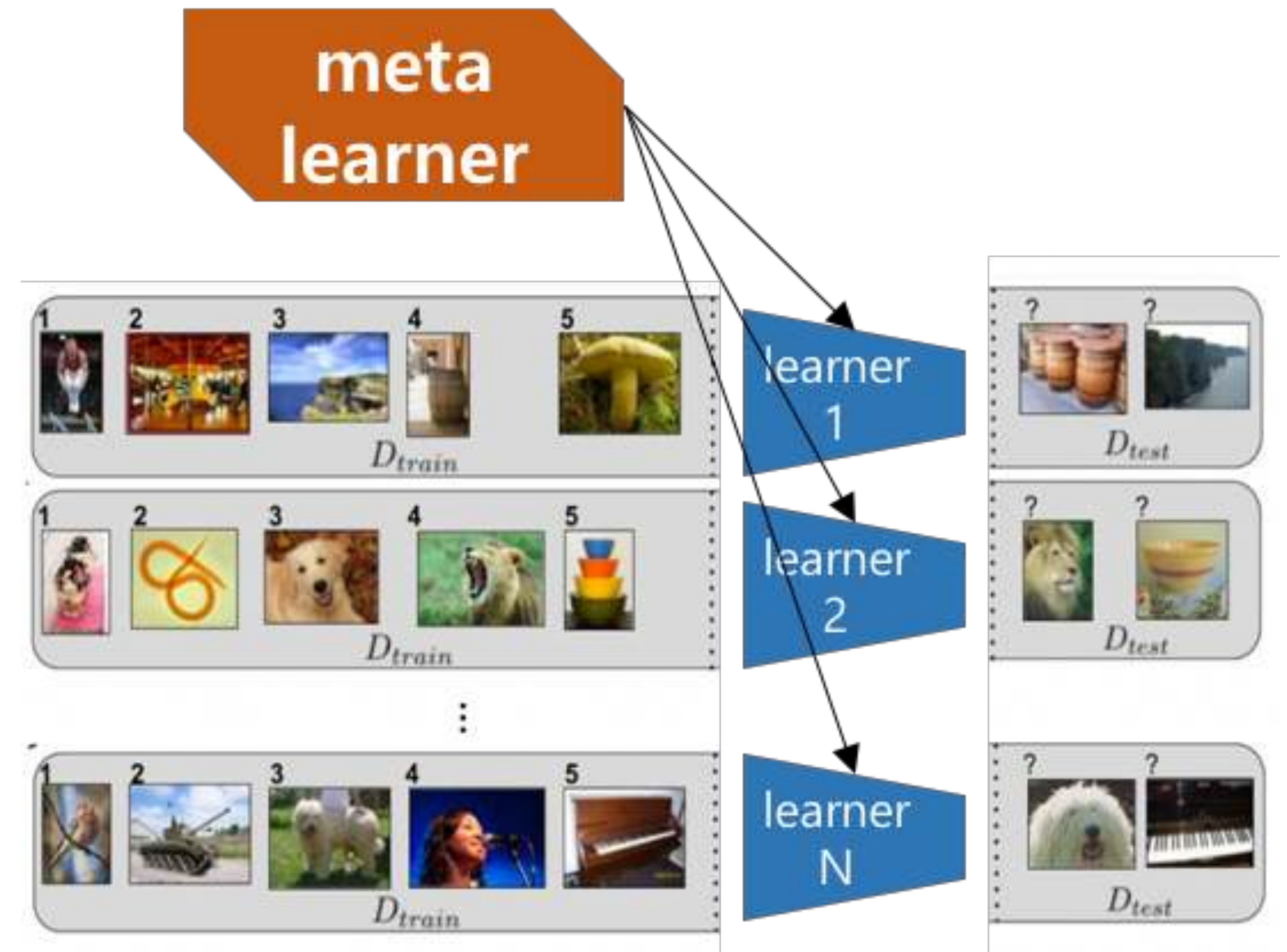
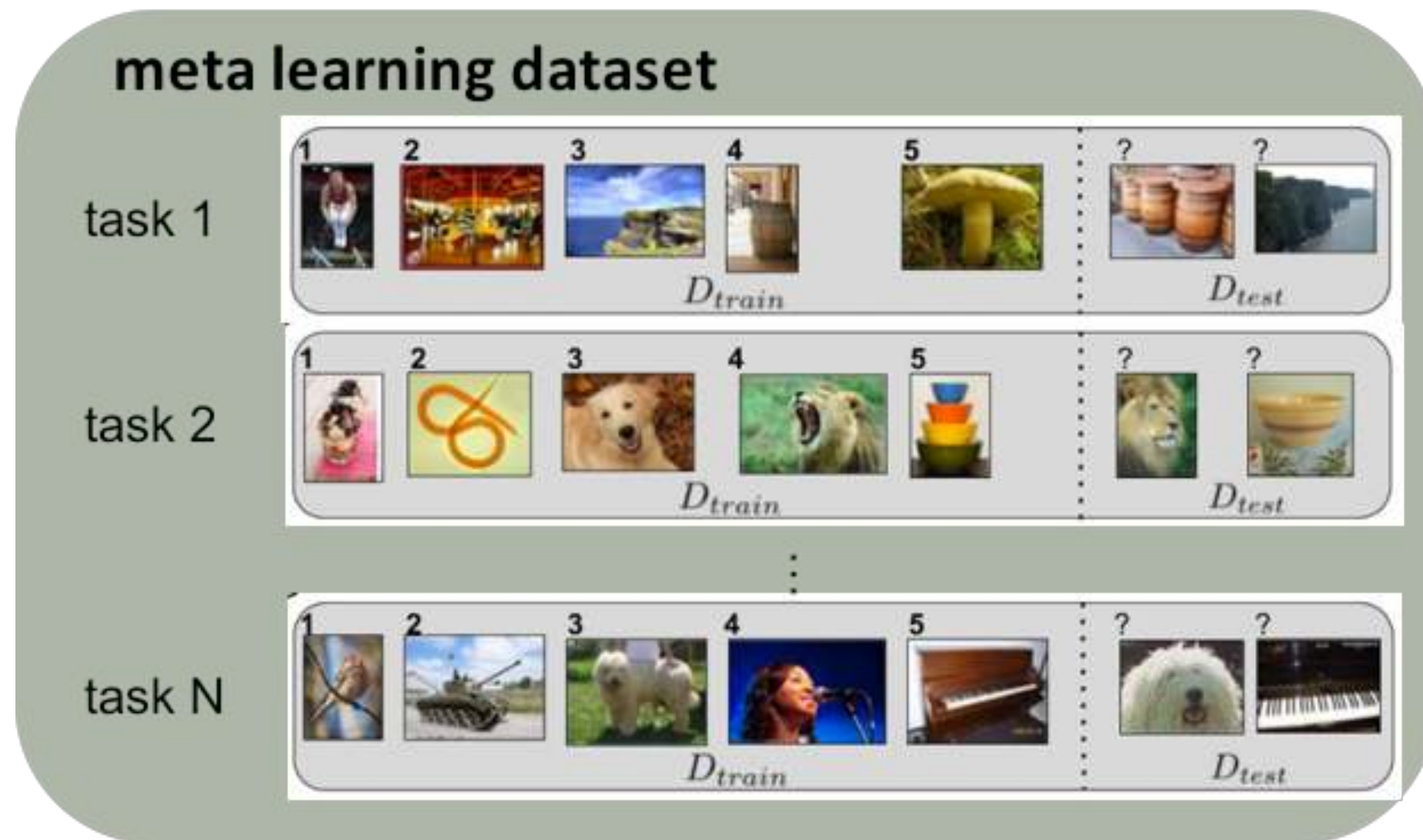
# 1. Problem Set-up

# Meta Supervised Learning



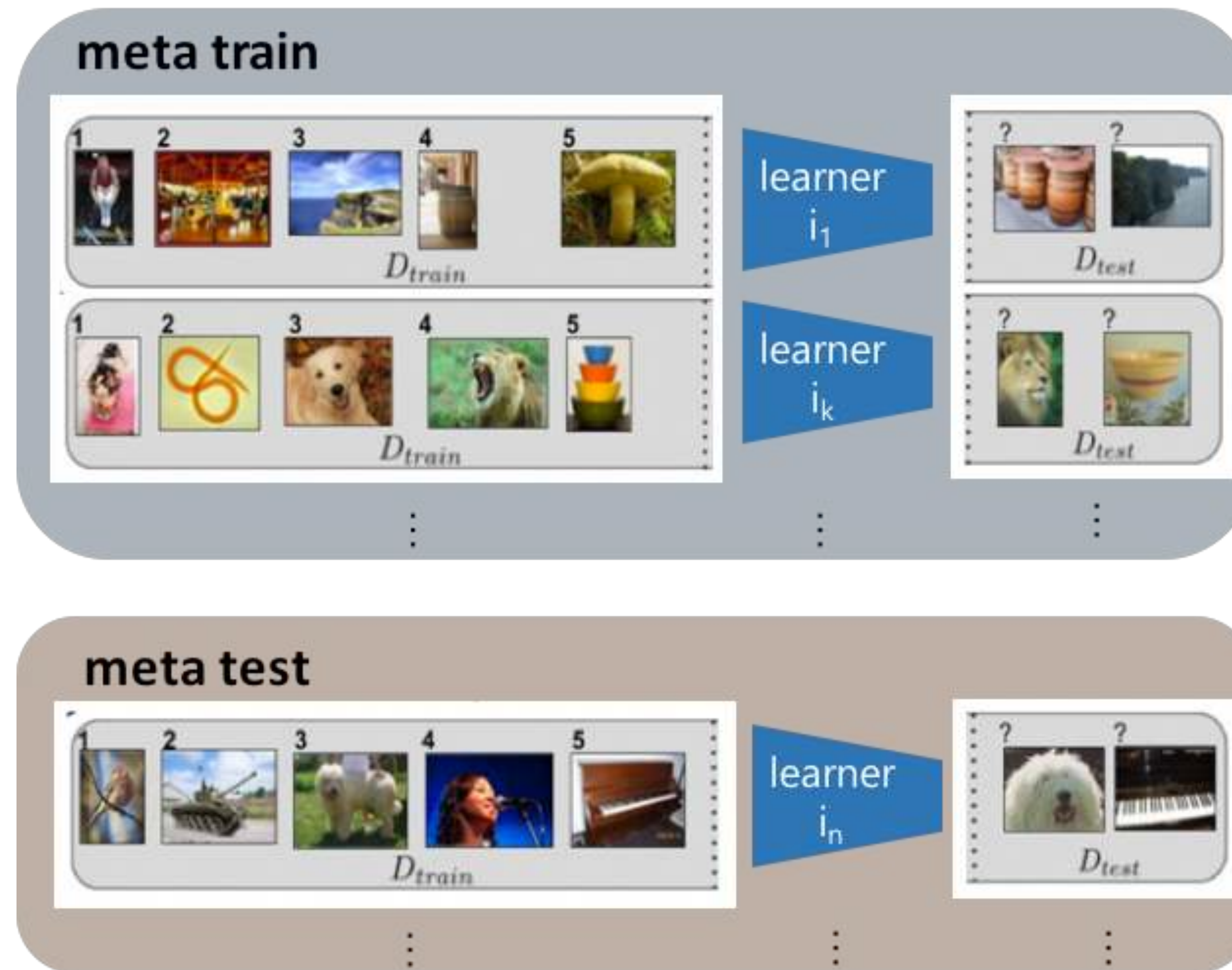
## 1. Problem Set-up

# Meta Supervised Learning



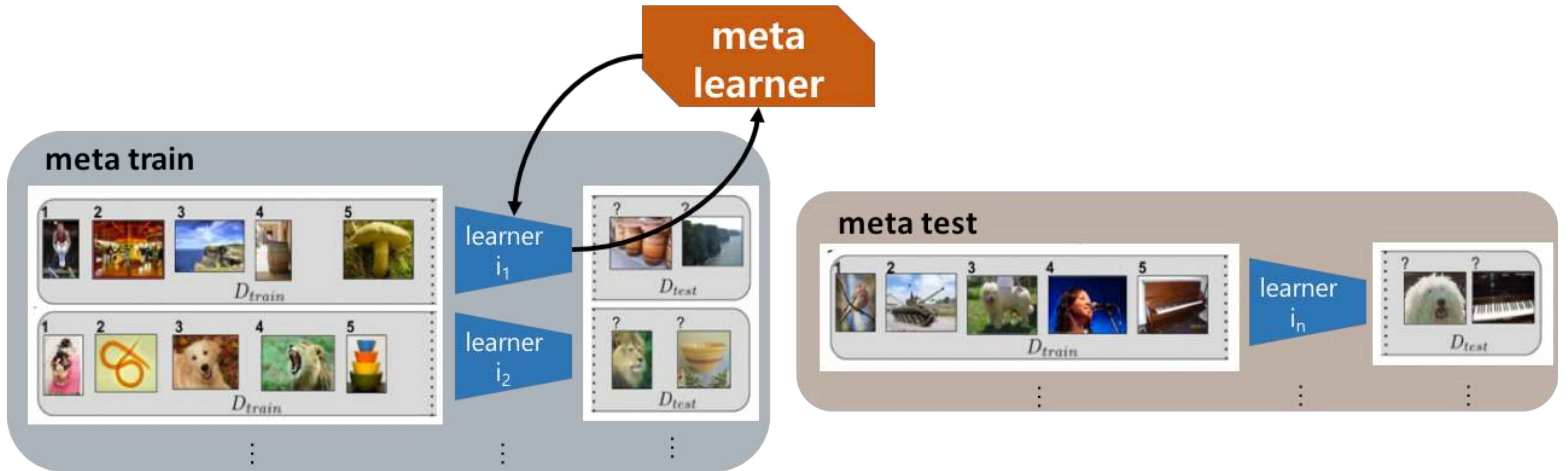
1. Problem Set-up

# Meta Supervised Learning



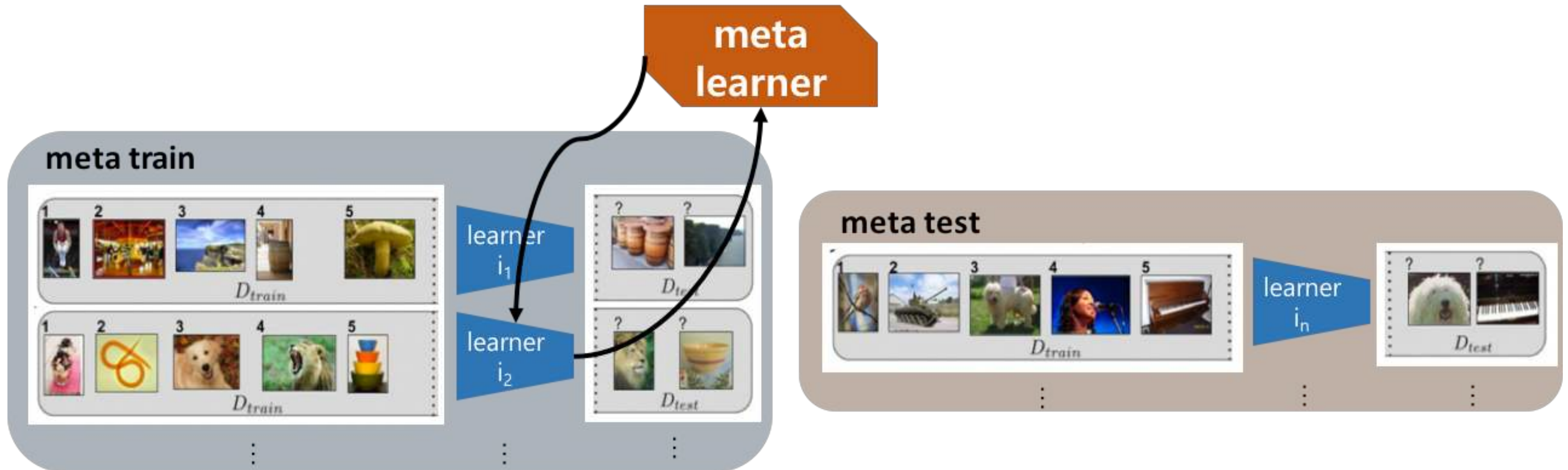
1. Problem Set-up

# Meta Supervised Learning



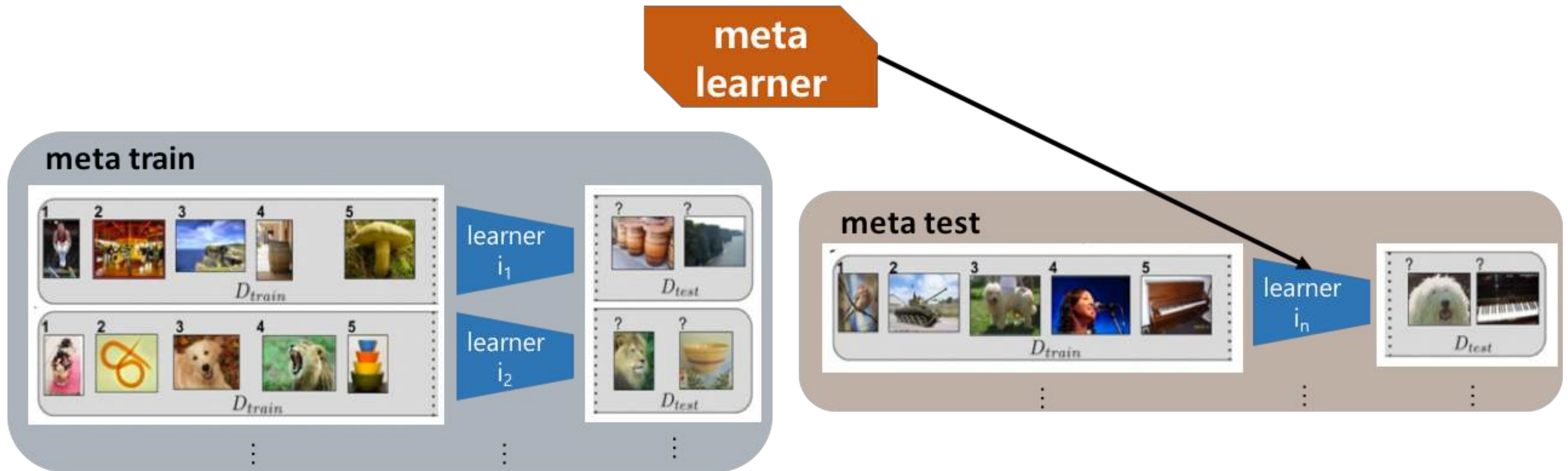
## 1. Problem Set-up

# Meta Supervised Learning



1. Problem Set-up

# Meta Supervised Learning



## 2. Prior Work

# Prior Work

- Learning an update function or update rule
  - LSTM optimizer  
(Learning to learn by gradient descent by gradient descent)
  - Meta LSTM optimizer  
(Optimization as a model for few-shot learning)
- Few shot (or meta) learning for specific tasks
  - Generative modeling (Neural Statistician)
  - Image classification (Matching Net., Prototypical Net.)
  - Reinforcement learning  
(Benchmarking deep reinforcement learning for continuous control)

## 2. Prior Work

# Prior Work

- Memory-augmented meta-learning
  - Memory-augmented models on many tasks  
(Meta-learning with memory-augmented neural networks)
- Parameter initialization of deep networks
  - Exploring sensitivity while maintaining Internal representation  
(Overcoming catastrophic forgetting in NN)
  - Data-dependent Initializer  
(Data-Dependent Initialization of CNN)
  - learned initialization  
(Gradient-based hyperparameter optimization through reversible learning)

### 3. Model-Agnostic Meta-Learning (MAML)

# Model-Agnostic Meta-Learning (MAML)

- Goal:
  - Quickly adapt to new tasks on distribution with **only small amount of data** and with **only a few gradient steps**, **even one gradient step**.
- Learner:
  - Learn a new task by using a single gradient step.
- Meta-learner:
  - Learn a **generalized parameter initialization** of model.

### 3. Model-Agnostic Meta-Learning (MAML)

# Characteristics of MAML

- The MAML learner's **weights are updated using the gradient**, rather than a learned update.
  - Not need additional parameters nor require a particular learner architecture.
- **Fast adaptability** through good parameter initialization
  - Explicitly optimizes to learn Internal representation (i.e. suitable for many tasks).
  - Maximizes sensitivity of new task losses to the model parameters.

### 3. Model-Agnostic Meta-Learning (MAML)

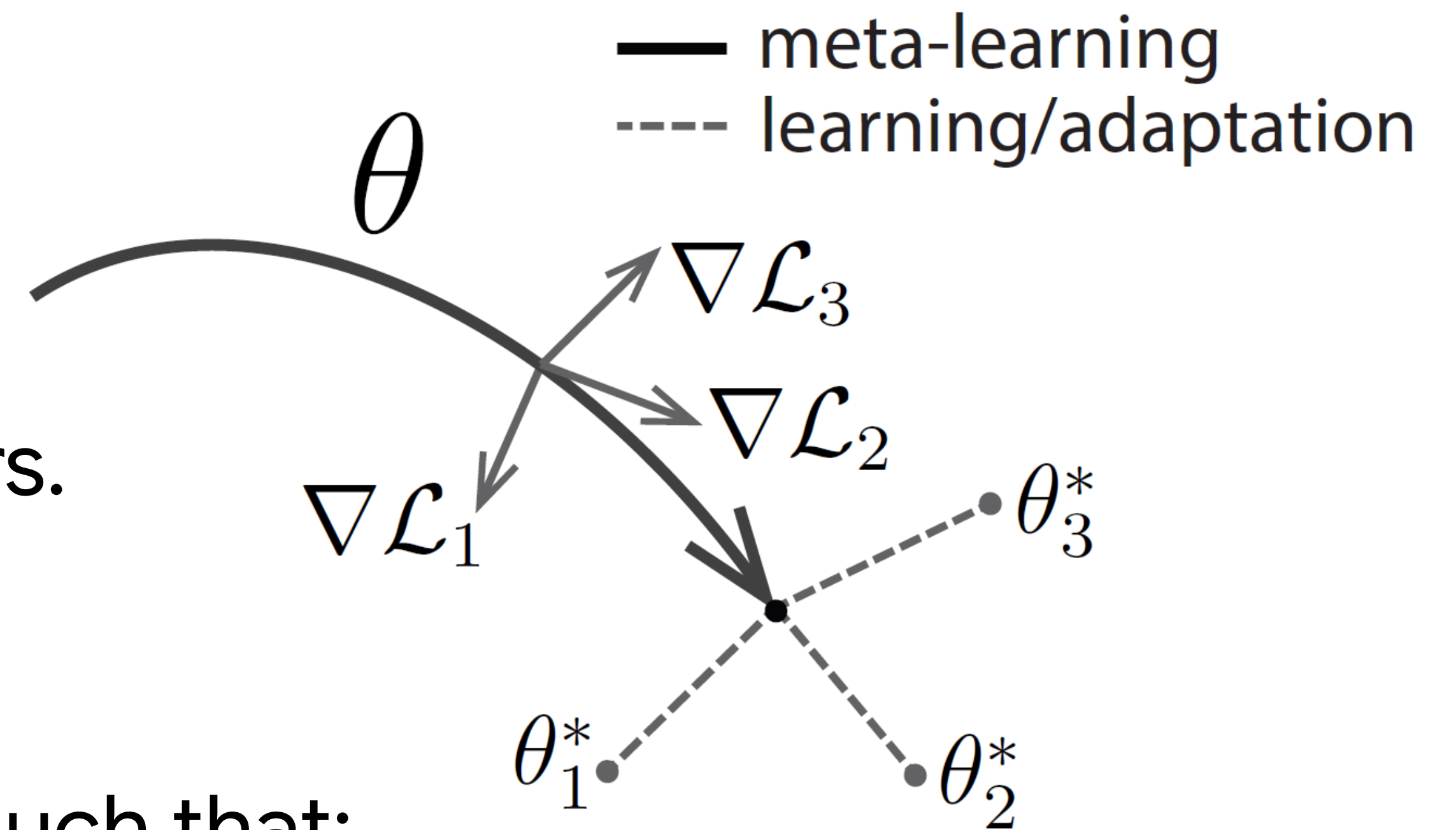
# Characteristics of MAML

- **Model-agnostic** (No matter whatever model is)
  - Classification & regression with differentiable losses, Policy gradient RL.
  - The model should be parameterized.
  - No other assumptions on the form of the model.
- **Task-agnostic** (No matter whatever task is)
  - Adopted all knowledge-transferable tasks.
  - No other assumption is required.

### 3. Model-Agnostic Meta-Learning (MAML)

## Intuition of MAML

- Some internal representations are **more transferrable** than others.
  - Desired model parameter set is  $\theta$  such that:  
Applying one (or a small # of) gradient step to  $\theta$  on a new task will produce maximally effective behavior.
- Find  $\theta$  that commonly decreases loss of each task **after adaptation**.



## 4. Approach

# Supervised Learning

- Notations
  - Model:  $f_{\theta}$  (Model-agnostic)
  - Task distribution:  $p(\mathcal{T})$
- For each Task:  $T = \{L(x_1, y_1, \dots, x_K, y_K), (x, y) \sim q\}$ 
  - Data distribution:  $q$  ( $K$  samples are drawn from  $q$ )
  - Loss function:  $\mathcal{L}$ 
    - MSE for Regression
    - Cross entropy for classification
    - Any other **differentiable** loss functions can be used.

## 4. Approach

# Supervised Learning

---

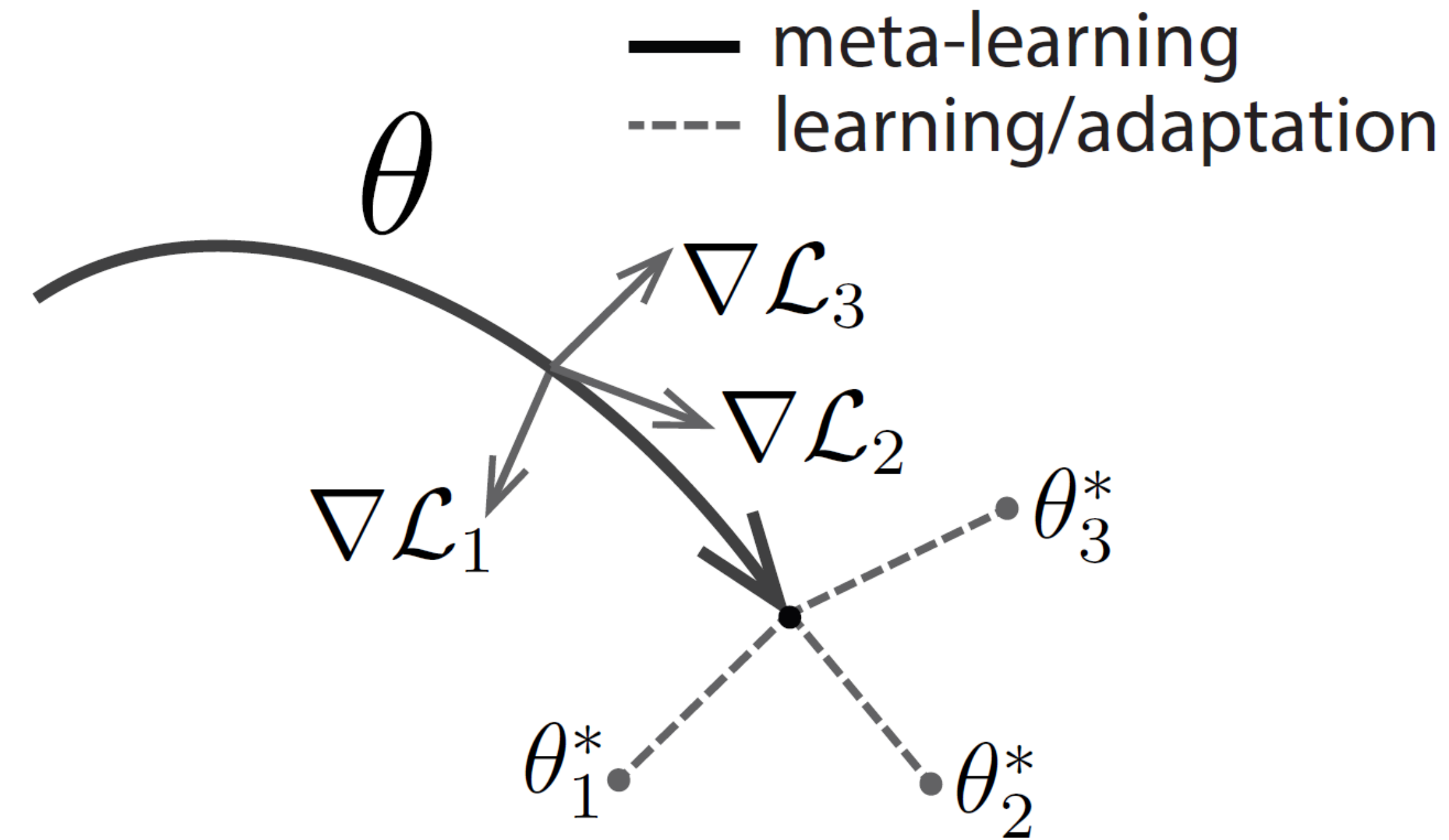
**Algorithm 2** MAML for Few-Shot Supervised Learning

---

**Require:**  $p(\mathcal{T})$ : distribution over tasks

**Require:**  $\alpha, \beta$ : step size hyperparameters

- 1: randomly initialize  $\theta$
  - 2: **while** not done **do**
  - 3:   Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$
  - 4:   **for all**  $\mathcal{T}_i$  **do**
  - 5:     Sample  $K$  datapoints  $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$  from  $\mathcal{T}_i$
  - 6:     Evaluate  $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$  using  $\mathcal{D}$  and  $\mathcal{L}_{\mathcal{T}_i}$
  - 7:     Compute adapted parameters with gradient descent:  
       $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
  - 8:     Sample datapoints  $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$  from  $\mathcal{T}_i$  for the meta-update
  - 9:   **end for**
  - 10:   Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$  using each  $\mathcal{D}'_i$  and  $\mathcal{L}_{\mathcal{T}_i}$
  - 11: **end while**
- 



#### 4. Approach

# Gradient of Gradient

- From line 10 in Algorithm 2,

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

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

#### 4. Approach

# Gradient of Gradient

- From line 10 in Algorithm 2,

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

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

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

( $\mathcal{L}$  is differentiable)

#### 4. Approach

# Gradient of Gradient

- From line 10 in Algorithm 2,

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

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

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

( $\mathcal{L}$  is differentiable)

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

#### 4. Approach

# Gradient of Gradient

- From line 10 in Algorithm 2,

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\text{Recall: } \theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))$$

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\mathcal{L} \text{ is differentiable})$$

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

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} (I - \alpha \nabla_{\theta}^2 \mathcal{L}_{\mathcal{T}_i}(f_{\theta})) \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$$

#### 4. Approach

# Gradient of Gradient

- From line 10 in Algorithm 2,

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\text{Recall: } \theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))$$

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\mathcal{L} \text{ is differentiable})$$

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

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \boxed{(I - \alpha \nabla_{\theta}^2 \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))} \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$$

Calculation of Hessian matrix is required.

#### 4. Approach

# Gradient of Gradient

- From line 10 in Algorithm 2,

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\text{Recall: } \theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))$$

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\mathcal{L} \text{ is differentiable})$$

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

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \boxed{(I - \alpha \nabla_{\theta}^2 \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))} \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$$

Calculation of Hessian matrix is required.

→ MAML suggest 1st order approximation.

#### 4. Approach

# 1st Order Approximation

- Update rule of MAML:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta})})$$

#### 4. Approach

# 1st Order Approximation

- Update rule of MAML:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i} \left( f_{\theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i} (f_{\theta})} \right)$$

This part needs calculating Hessian.  
Hessian makes MAML be slow.

#### 4. Approach

# 1st Order Approximation

- Update rule of MAML:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta})})$$

- Update rule of MAML with 1st order approximation:

$$\delta \leftarrow \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$$

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \delta})$$

#### 4. Approach

# 1st Order Approximation

- Update rule of MAML:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta})})$$

- Update rule of MAML with 1st order approximation:

$$\delta \leftarrow \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta}) \quad (\text{Regard } \delta \text{ as constant})$$

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \delta})$$

#### 4. Approach

## Recall: Gradient of Gradient

- From line 10 in Algorithm 2,

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\text{Recall: } \theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))$$

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) \quad (\mathcal{L} \text{ is differentiable})$$

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

$$= \theta - \beta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \boxed{(I - \alpha \nabla_{\theta}^2 \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))} \nabla_{\theta'_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$$

In 1st order approximation,  
we regard this as identity matrix  $I$ .

## 4. Approach

# Reinforcement Learning

- Notations
  - Policy of an agent:  $f_\theta$  such that  $a_t \sim f_\theta(x_t)$
  - Task distribution:  $p(\mathcal{T})$
- For each Task:  $T = \{L(e_1, e_2, \dots, e_K), x_1 \sim q(x), x_{t+1} \sim q(x|x_t, a_t), H\}$ 
  - Episode:  $e = (x_1, a_1, \dots, x_H, a_H)$
  - Transition distribution:  $q$  ( $K$  trajectories are drawn from  $q$  and  $f$ )
  - Episode length:  $H$
  - Loss function:  $\mathcal{L}$  = expectation of sum of rewards

## 4. Approach

# Reinforcement Learning

---

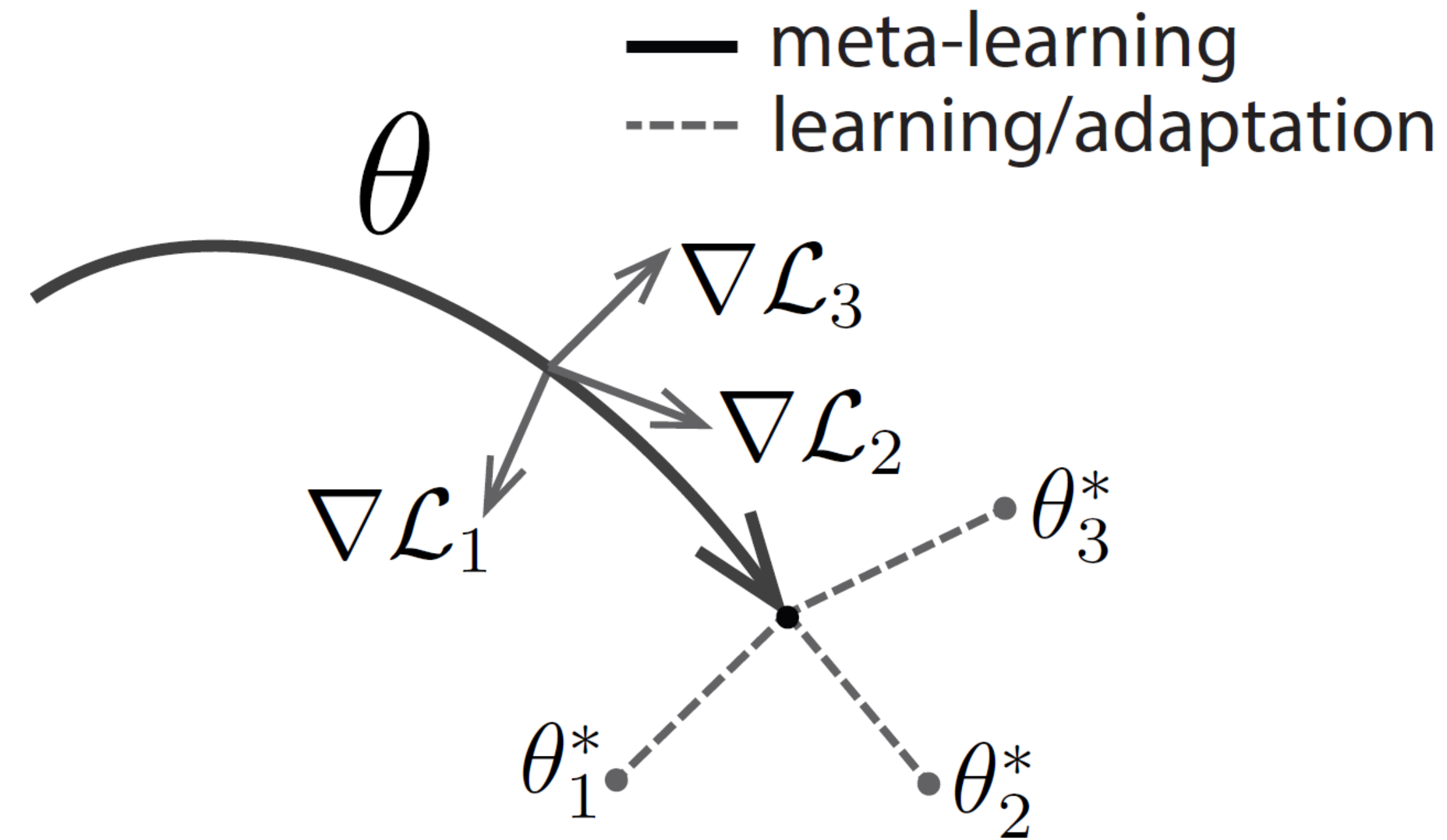
### Algorithm 3 MAML for Reinforcement Learning

---

**Require:**  $p(\mathcal{T})$ : distribution over tasks

**Require:**  $\alpha, \beta$ : step size hyperparameters

- 1: randomly initialize  $\theta$
  - 2: **while** not done **do**
  - 3:   Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$
  - 4:   **for all**  $\mathcal{T}_i$  **do**
  - 5:     Sample  $K$  trajectories  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, \dots, \mathbf{x}_H)\}$  using  $f_\theta$  in  $\mathcal{T}_i$
  - 6:     Evaluate  $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$  using  $\mathcal{D}$  and  $\mathcal{L}_{\mathcal{T}_i}$
  - 7:     Compute adapted parameters with gradient descent:  
     $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
  - 8:     Sample trajectories  $\mathcal{D}'_i = \{(\mathbf{x}_1, \mathbf{a}_1, \dots, \mathbf{x}_H)\}$  using  $f_{\theta'_i}$  in  $\mathcal{T}_i$
  - 9:   **end for**
  - 10:   Update  $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$  using each  $\mathcal{D}'_i$  and  $\mathcal{L}_{\mathcal{T}_i}$
  - 11: **end while**
- 



## 5. Experiment

# Experiment on Few-Shot Classification

- Omniglot (Lake et al., 2012)
  - 50 different alphabets, 1623 characters.
  - 20 instances for each characters were drawn by 20 different people.
  - 1200 for training, 423 for test.
- Mini-Imagenet (Ravi & Larochelle, 2017)
  - Classes for each set: train=64, validation=12, test=24.

## 5. Experiment

# Results of Few-Shot Classification

- MAML outperforms state-of-arts algorithms.

	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Omniglot (Lake et al., 2011)				
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
<b>MAML, no conv (ours)</b>	<b>89.7 ± 1.1%</b>	<b>97.5 ± 0.6%</b>	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
<b>MAML (ours)</b>	<b>98.7 ± 0.4%</b>	<b>99.9 ± 0.1%</b>	<b>95.8 ± 0.3%</b>	<b>98.9 ± 0.2%</b>

## 5. Experiment

# Effect of 1st Order Approximation

- In 1st order approximation, computation is roughly 33% better.
- Performances are similar.

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy	
	1-shot	5-shot
fine-tuning baseline	28.86 $\pm$ 0.54%	49.79 $\pm$ 0.79%
nearest neighbor baseline	41.08 $\pm$ 0.70%	51.04 $\pm$ 0.65%
matching nets (Vinyals et al., 2016)	43.56 $\pm$ 0.84%	55.31 $\pm$ 0.73%
meta-learner LSTM (Ravi & Larochelle, 2017)	43.44 $\pm$ 0.77%	60.60 $\pm$ 0.71%
<b>MAML, first order approx. (ours)</b>	<b>48.07 <math>\pm</math> 1.75%</b>	<b>63.15 <math>\pm</math> 0.91%</b>
<b>MAML (ours)</b>	<b>48.70 <math>\pm</math> 1.84%</b>	<b>63.11 <math>\pm</math> 0.92%</b>

## 5. Experiment

# Effect of 1st Order Approximation

- In 1st order approximation, computation is roughly 33% better.
- Performances are similar.

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy	
	1-shot	5-shot
fine-tuning baseline	28.86 ± 0.54%	49.79 ± 0.79%
nearest neighbor baseline	41.08 ± 0.70%	51.04 ± 0.65%
matching nets (Vinyals et al., 2016)	43.56 ± 0.84%	55.31 ± 0.73%
meta-learner LSTM (Ravi & Larochelle, 2017)	43.44 ± 0.77%	60.60 ± 0.71%
<b>MAML, first order approx. (ours)</b>	<b>48.07 ± 1.75%</b>	<b>63.15 ± 0.91%</b>
<b>MAML (ours)</b>	<b>48.70 ± 1.84%</b>	<b>63.11 ± 0.92%</b>

Performances are not that different.

## 5. Experiment

# Effect of 1st Order Approximation

- In 1st order approximation, computation is roughly 33% better.
- Performances are similar.

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy	
	1-shot	5-shot
fine-tuning baseline	28.86 ± 0.54%	49.79 ± 0.79%
nearest neighbor baseline	41.08 ± 0.70%	51.04 ± 0.65%
matching nets (Vinyals et al., 2016)	43.56 ± 0.84%	55.31 ± 0.73%
meta-learner LSTM (Ravi & Larochelle, 2017)	43.44 ± 0.77%	60.60 ± 0.71%
<b>MAML, first order approx. (ours)</b>	<b>48.07 ± 1.75%</b>	<b>63.15 ± 0.91%</b>
<b>MAML (ours)</b>	<b>48.70 ± 1.84%</b>	<b>63.11 ± 0.92%</b>

Some performance is even better.

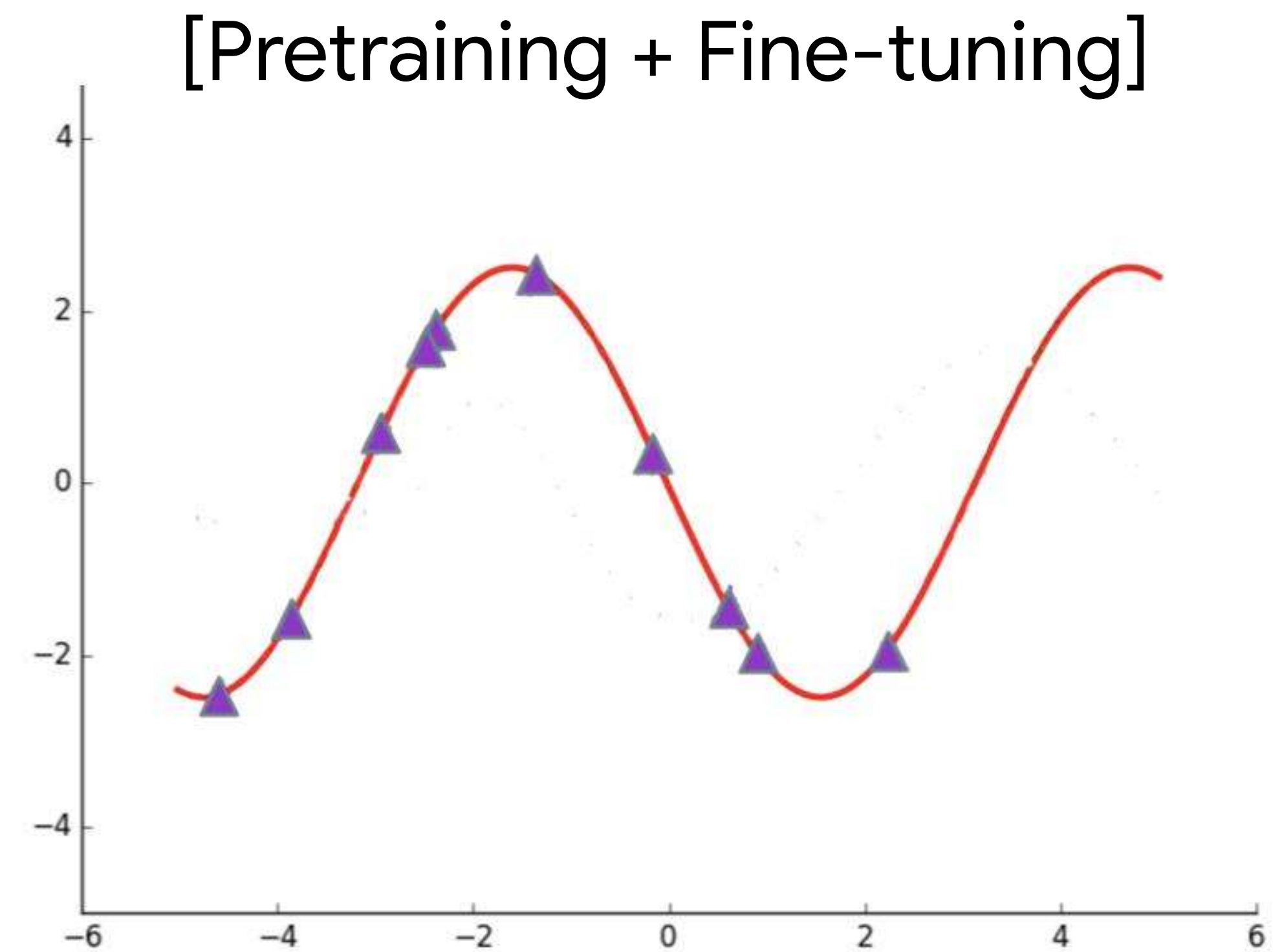
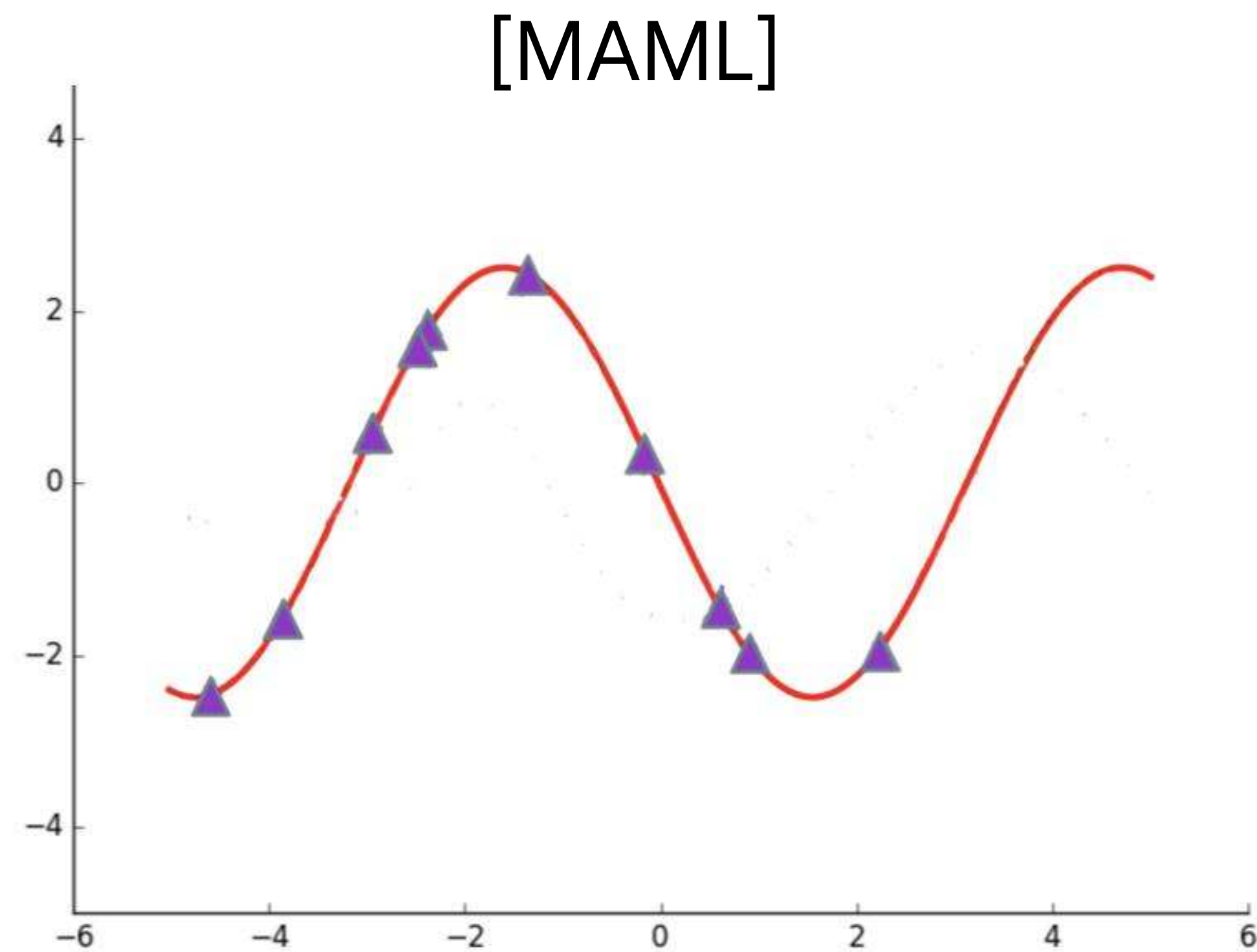
## 5. Experiment

# Experiments on Regression

- Sinusoid function:
  - Amplitude ( $A$ ) and phase ( $\phi$ ) are varied between tasks
    - $A$  in  $[0.1, 0.5]$
    - $\phi$  in  $[0, \pi]$
  - $x$  in  $[-5.0, 5.0]$
- Loss function: Mean Squared Error (MSE)
- Regressor: 2 hidden layers with 40 units and ReLU
- Training
  - Use **only 1 gradient step** for learner
  - $K = 5$  or 10 example (5-shot learning or 10-shot learning)
  - Fixed step size ( $\alpha=0.01$ ) for Adam optimizer.

## 5. Experiment

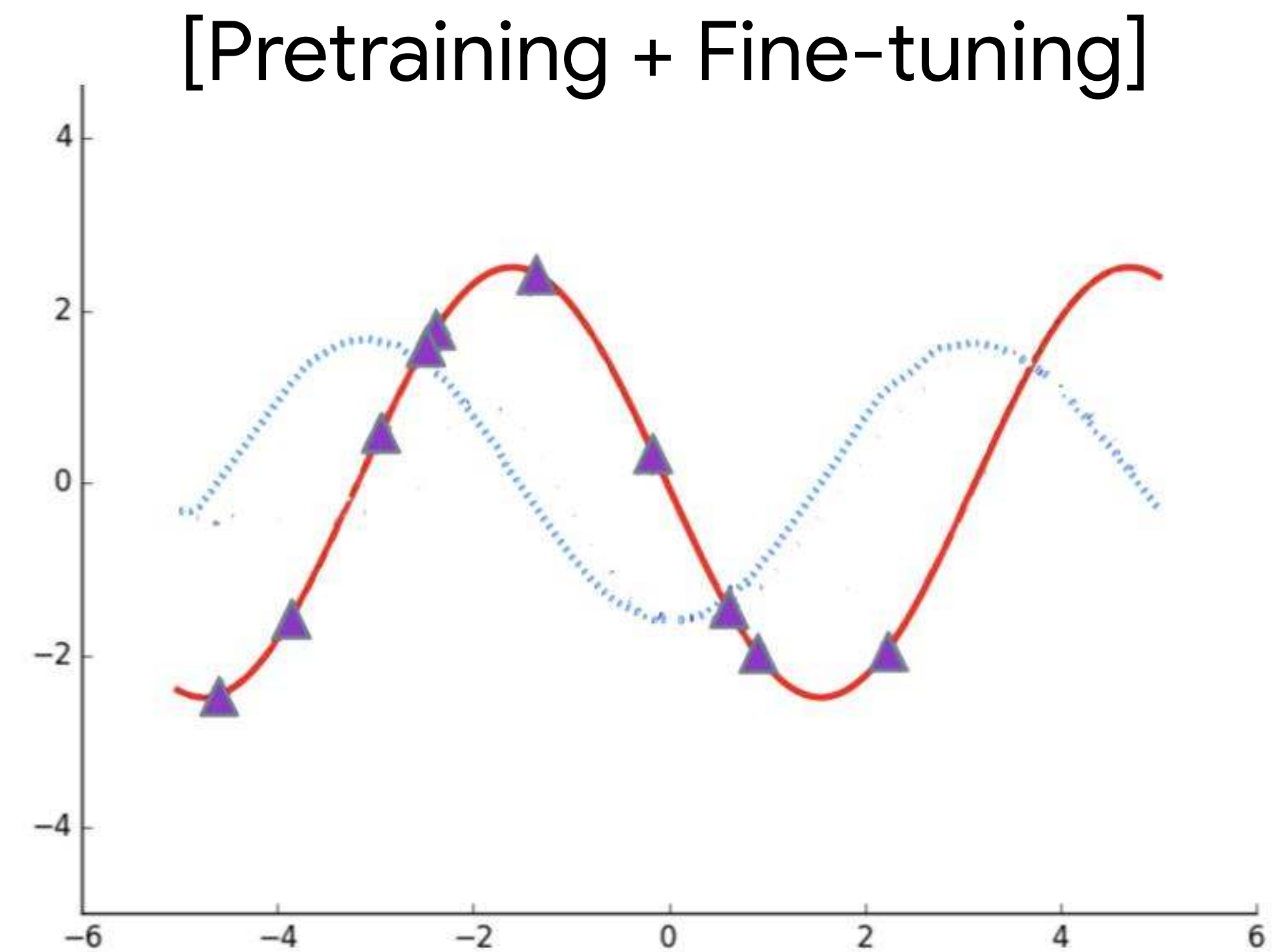
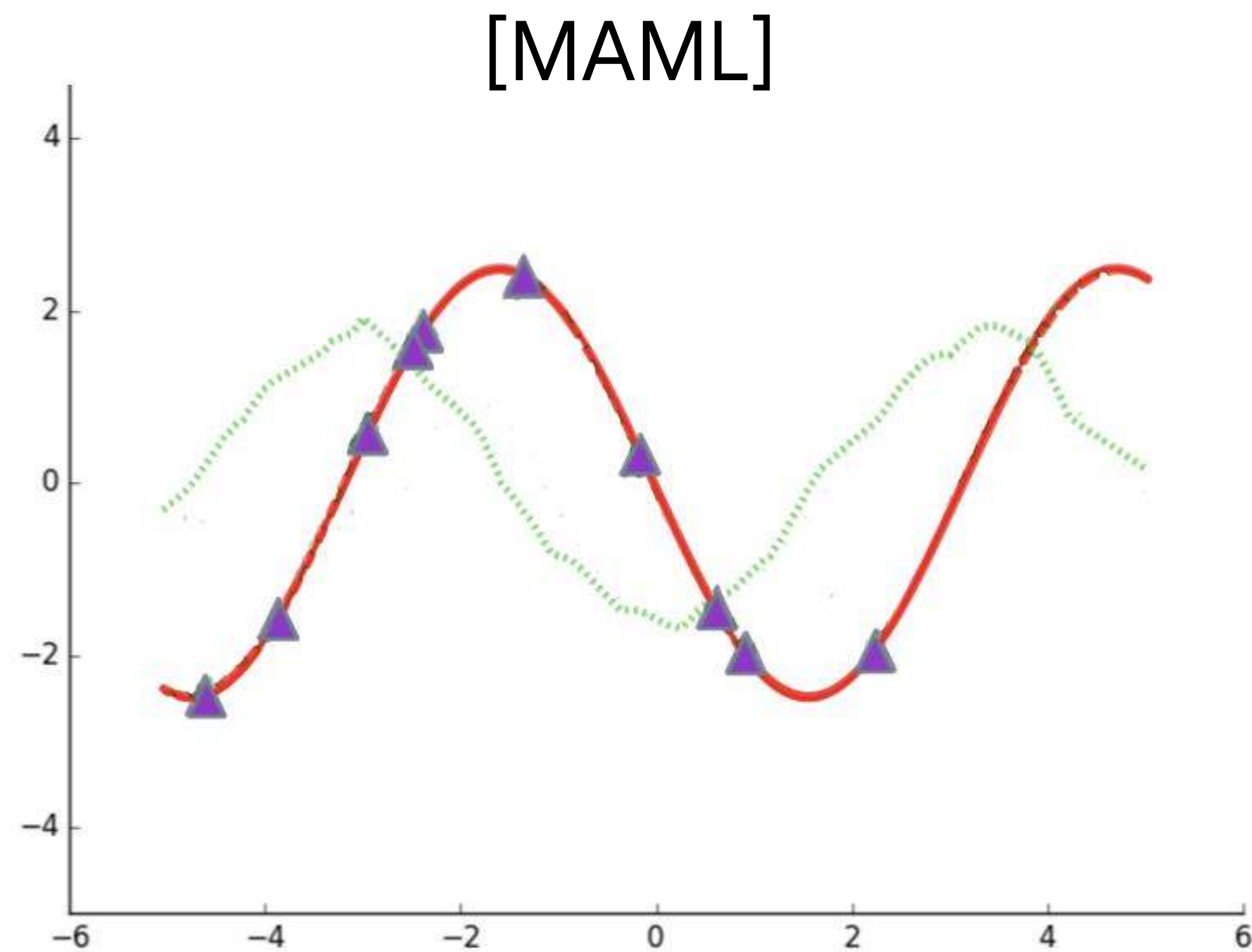
# Results of 10-Shot Learning Regression



The red line is ground truth.  
Fit this sine function with only few (10) samples.

## 5. Experiment

# Results of 10-Shot Learning Regression



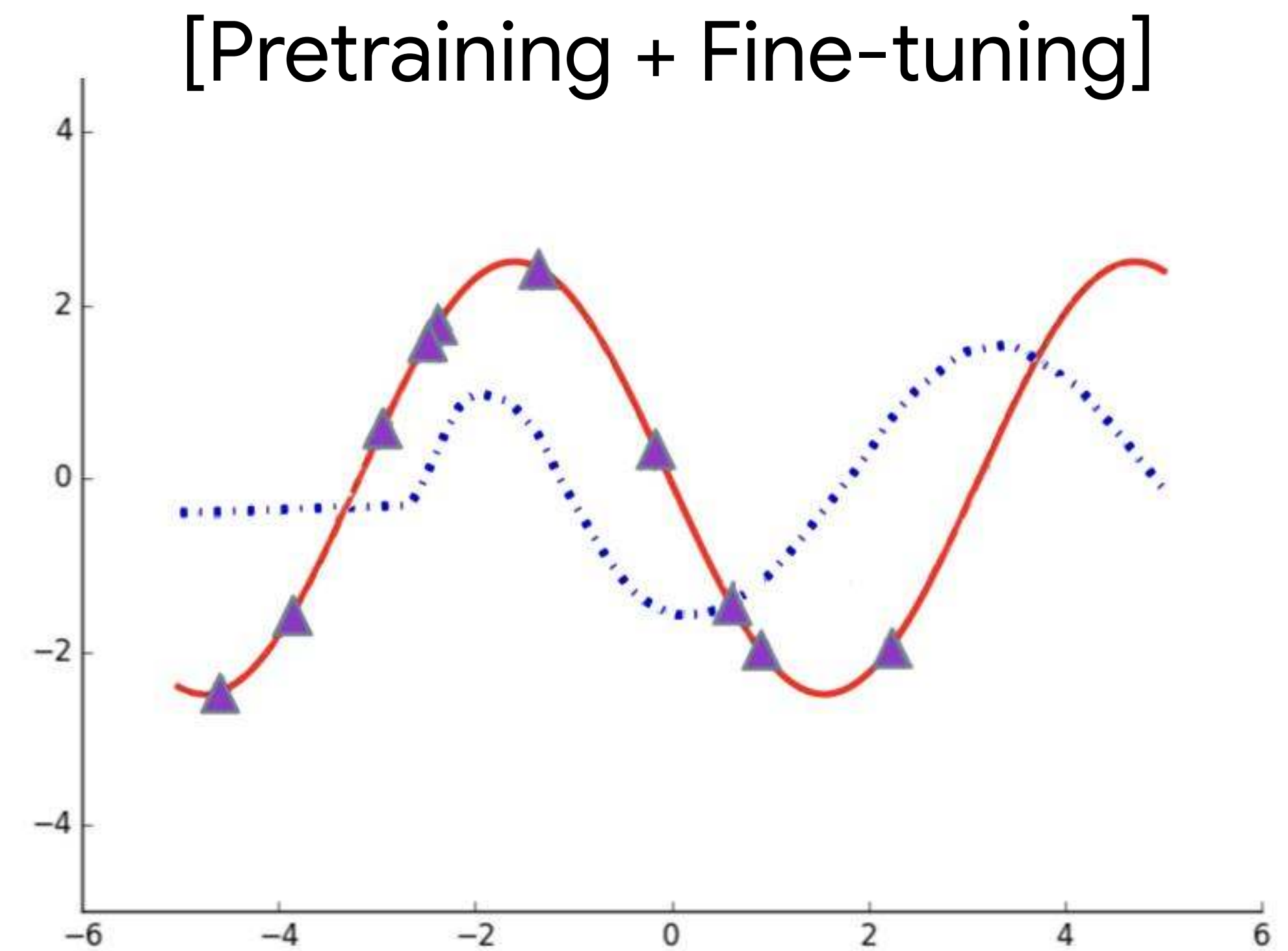
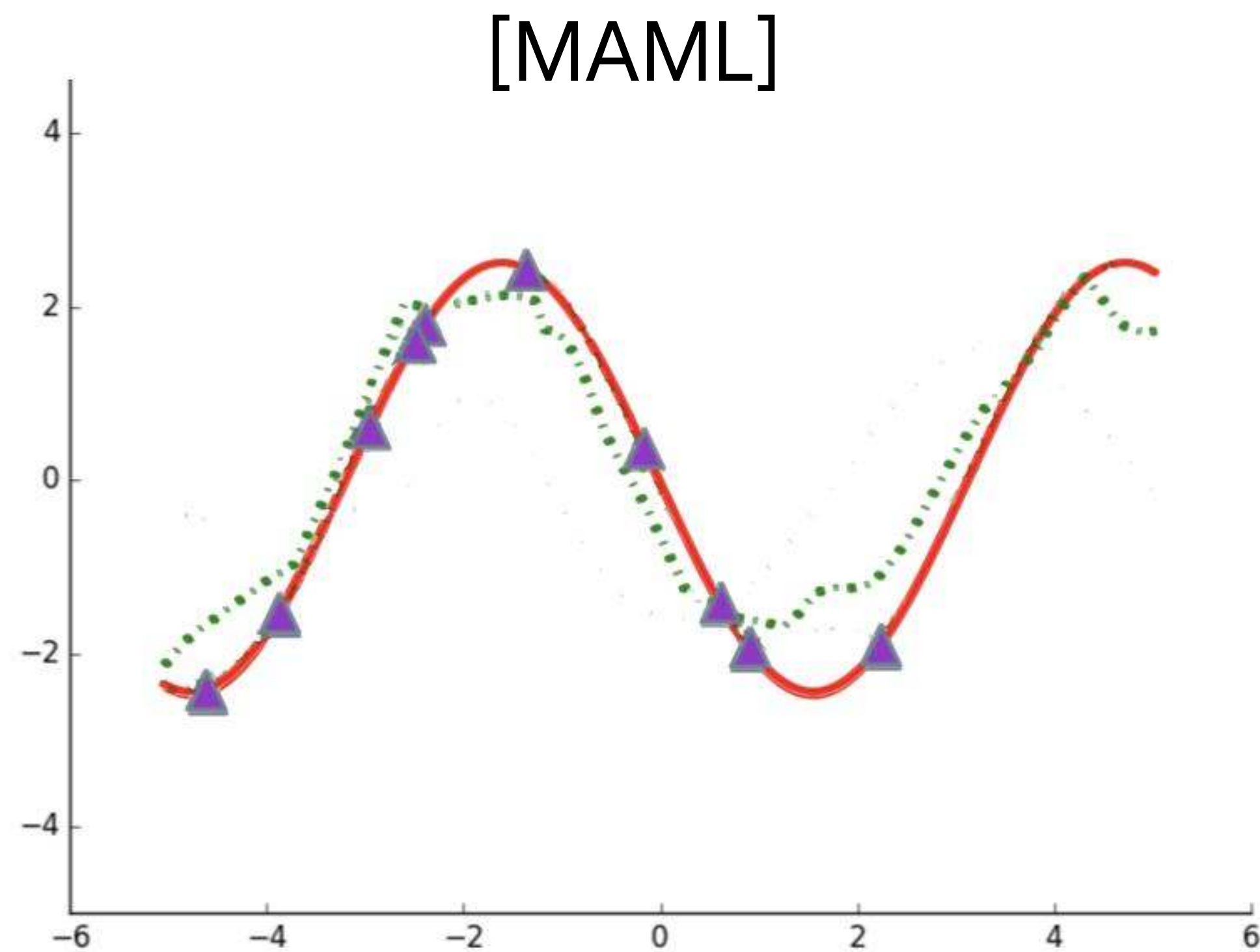
Above plots are the pre-trained function of two models.

(The prediction of meta-parameter of MAML,

The prediction of co-learned parameter of vanilla multi-task learning)

## 5. Experiment

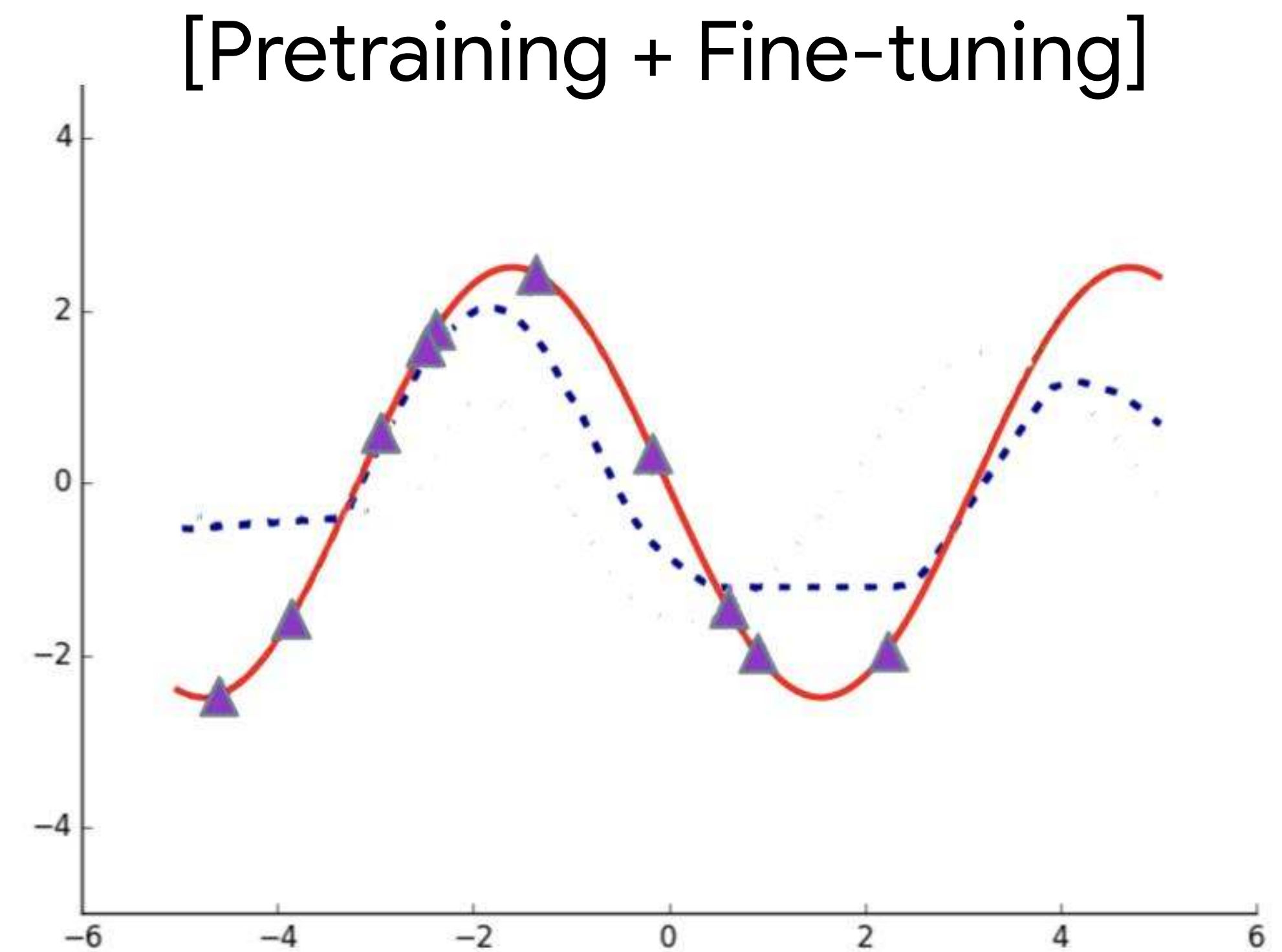
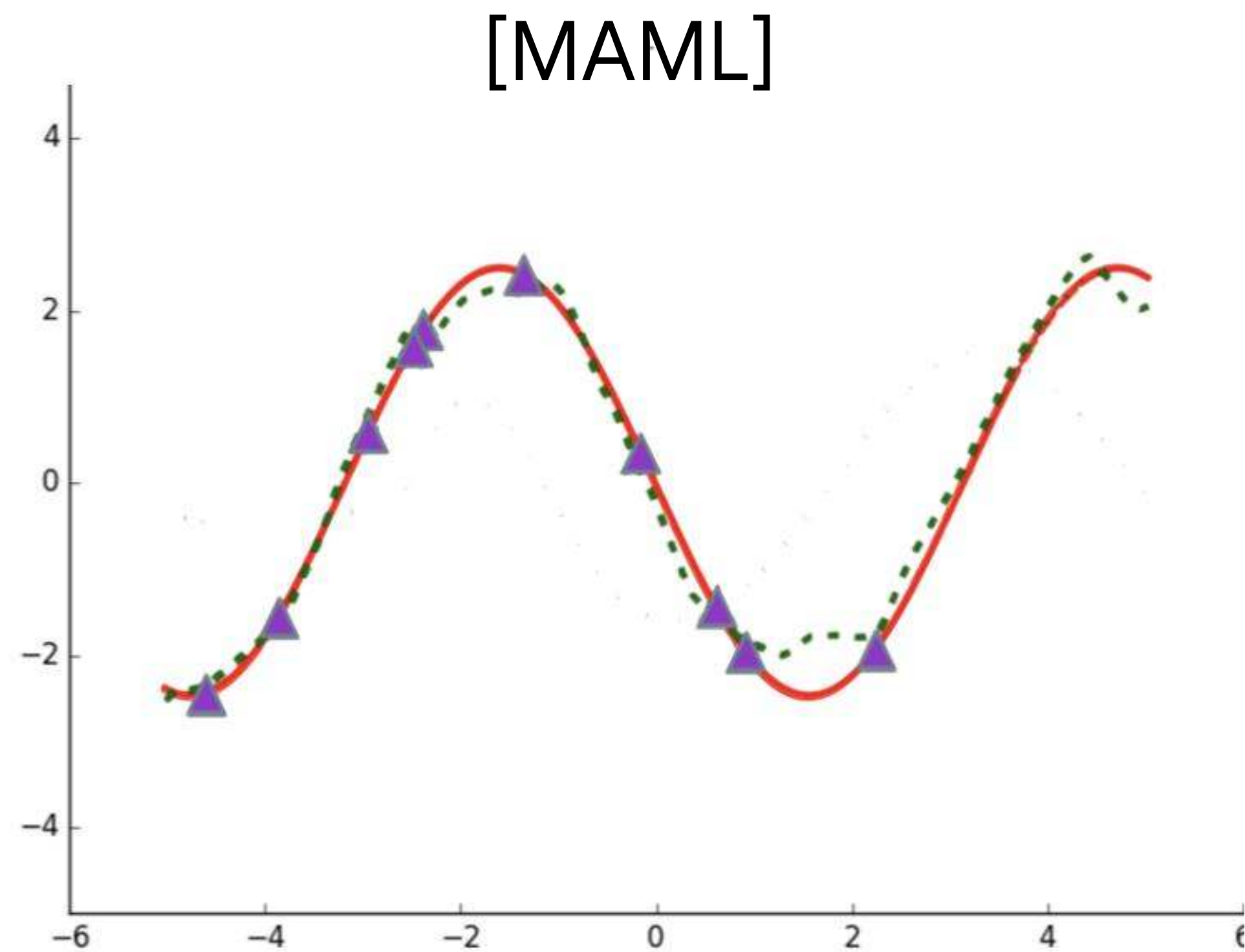
# Results of 10-Shot Learning Regression



After 1 gradient step update.

## 5. Experiment

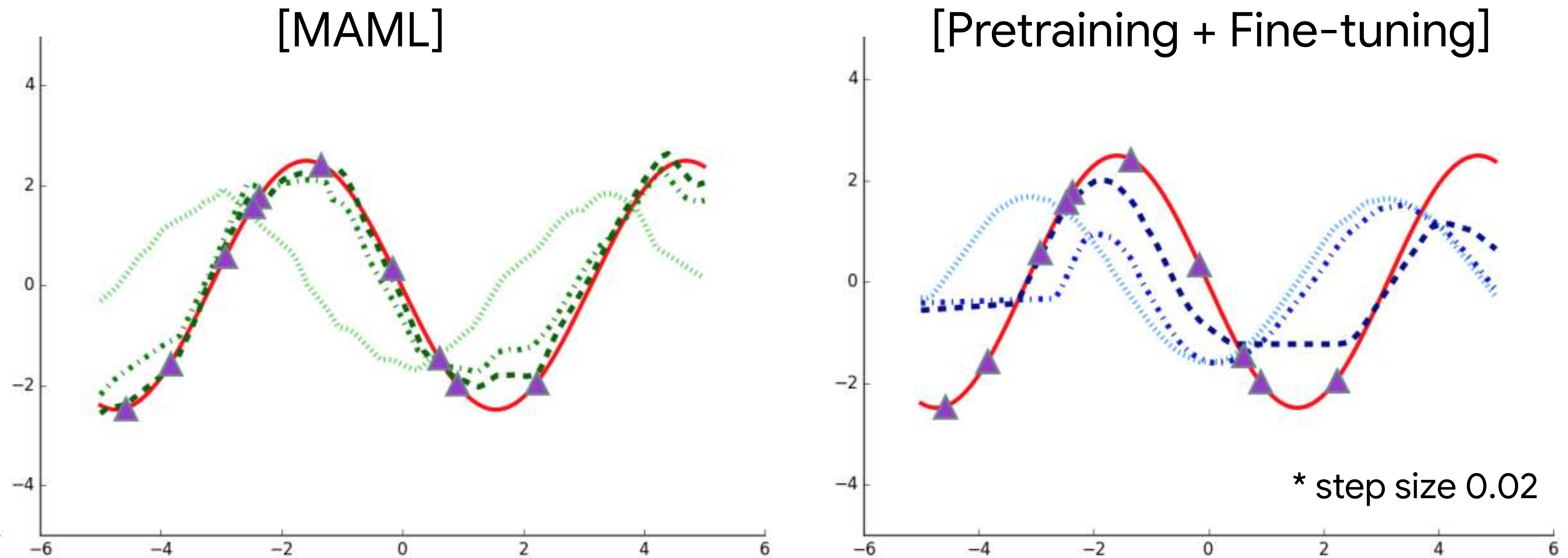
# Results of 10-Shot Learning Regression



After 10 gradient step update.

## 5. Experiment

# Results of 10-Shot Learning Regression

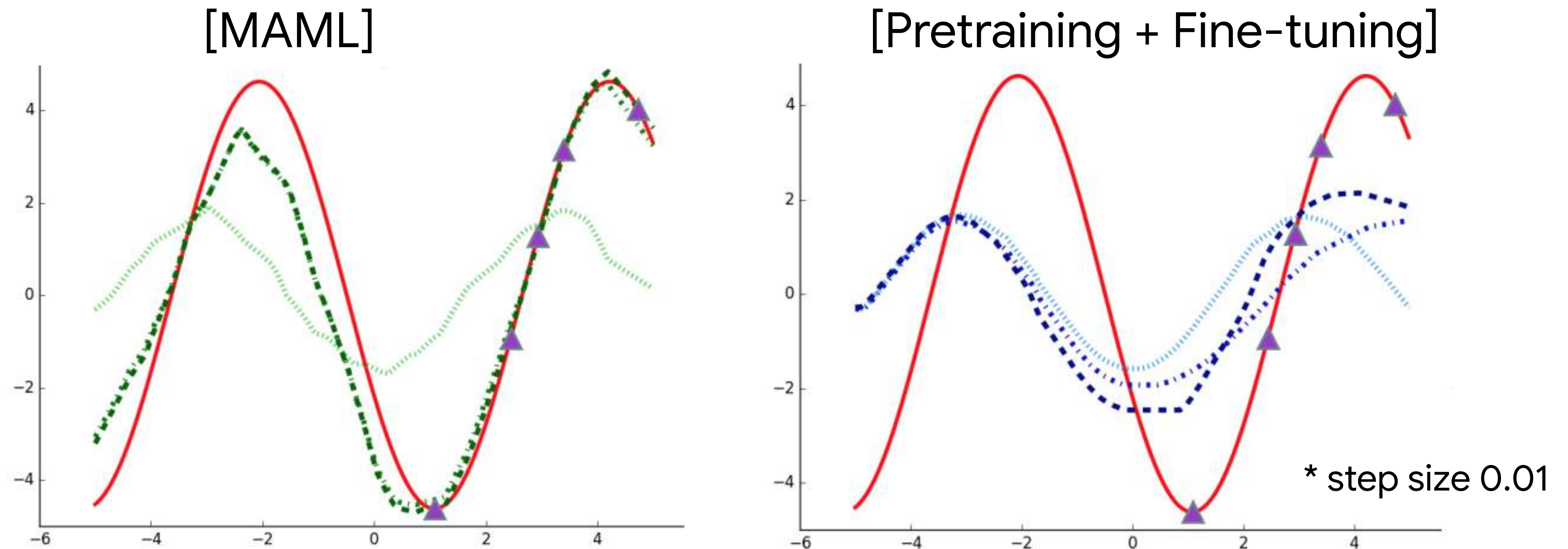


In the case of MAML, they quickly adapted to new samples.  
But, adaptation failed in the case of pretraining model.

..... pre-update    ..... 1 grad step    ..... 10 grad steps    — ground truth    ▲ ▲ used for grad    ..... pre-update    ..... 1 grad step    ..... 10 grad steps

## 5. Experiment

# Results of 5-Shot Learning Regression

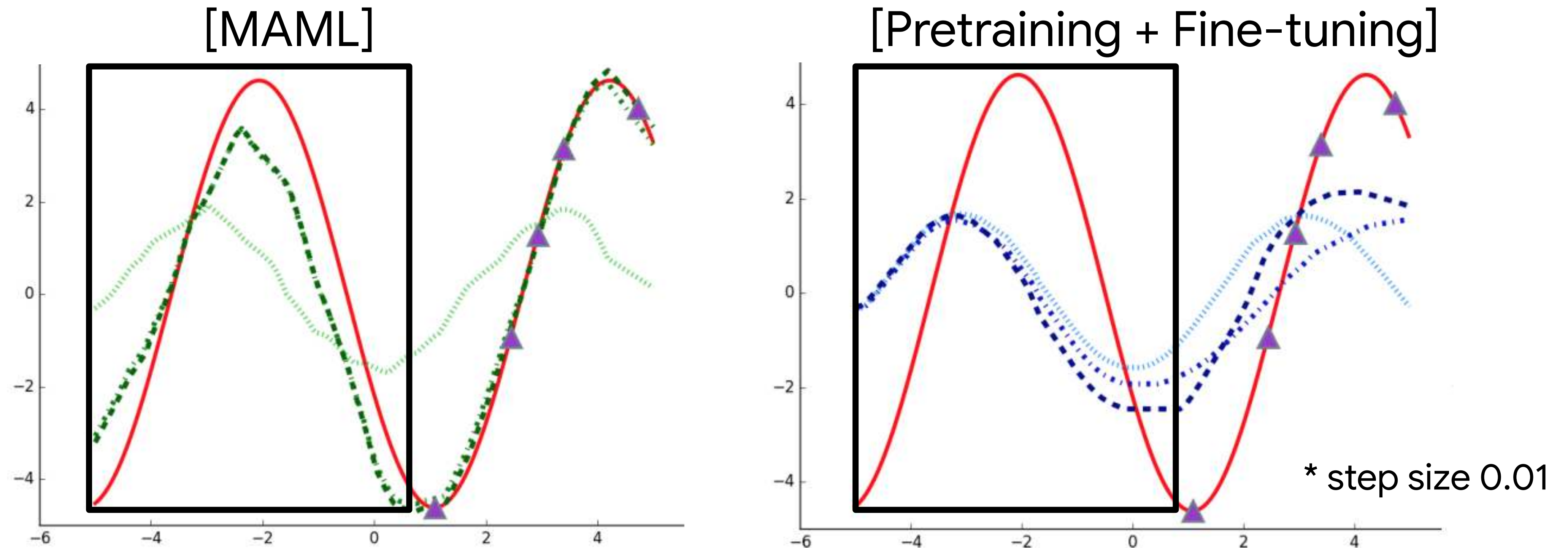


In the 5-shot learning, the difference is pervasive.

..... pre-update    ..... 1 grad step    ..... 10 grad steps    — ground truth    ▲ ▲ used for grad    ..... pre-update    ..... 1 grad step    ..... 10 grad steps

## 5. Experiment

# Results of 5-Shot Learning Regression

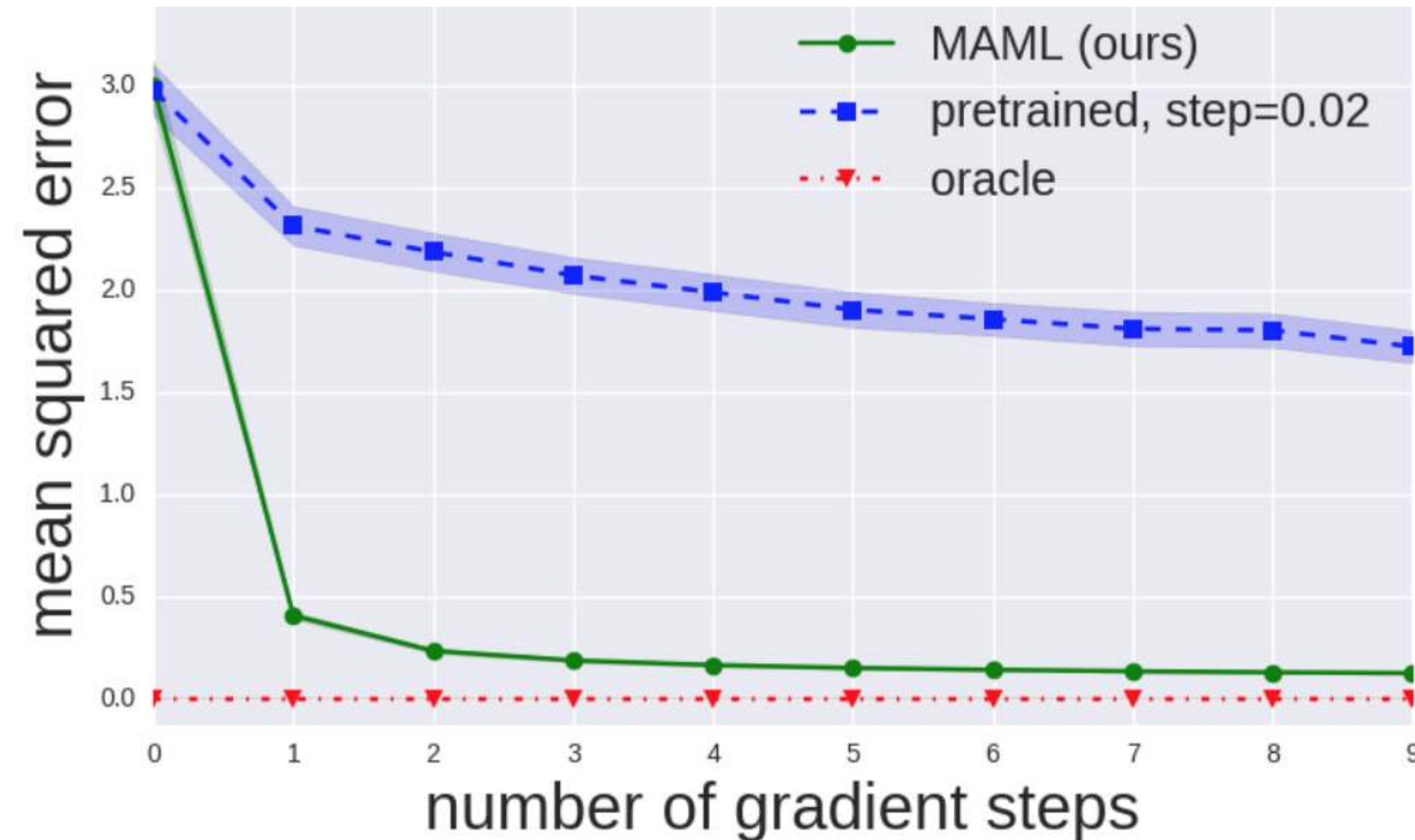


In the 5-shot learning, the difference is pervasive.  
Good predictions are also made for ranges not particularly seen.

..... pre-update    ..... 1 grad step    -.-.- 10 grad steps    — ground truth    ▲ ▲ used for grad    ..... pre-update    ..... 1 grad step    -.-.- 10 grad steps

## 5. Experiment

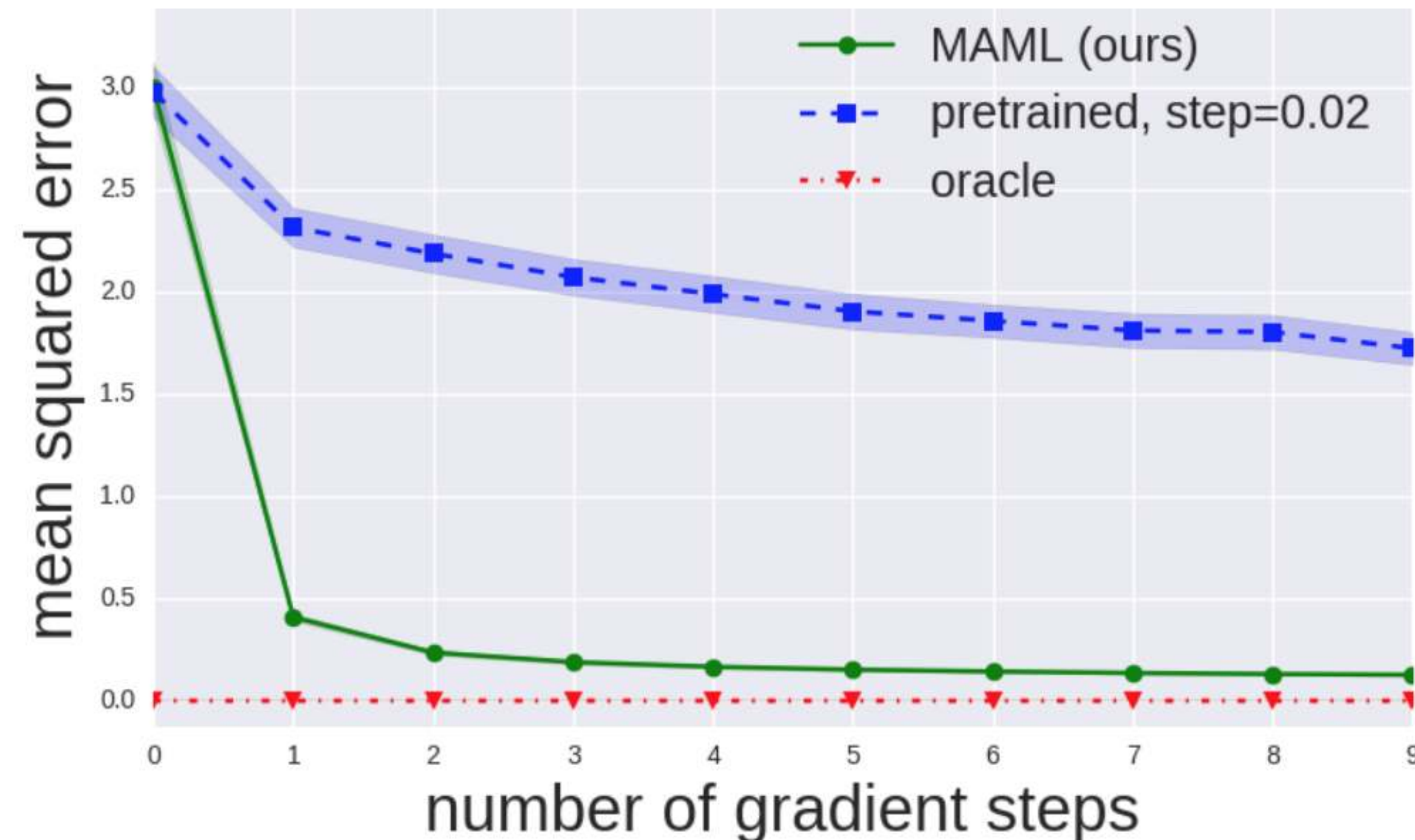
# MAML Needs Only One Gradient Step



Vanilla pretrained model adapted slowly,  
but, the MAML method quickly adapted **even in one gradient step**.

## 5. Experiment

# MAML Needs Only One Gradient Step

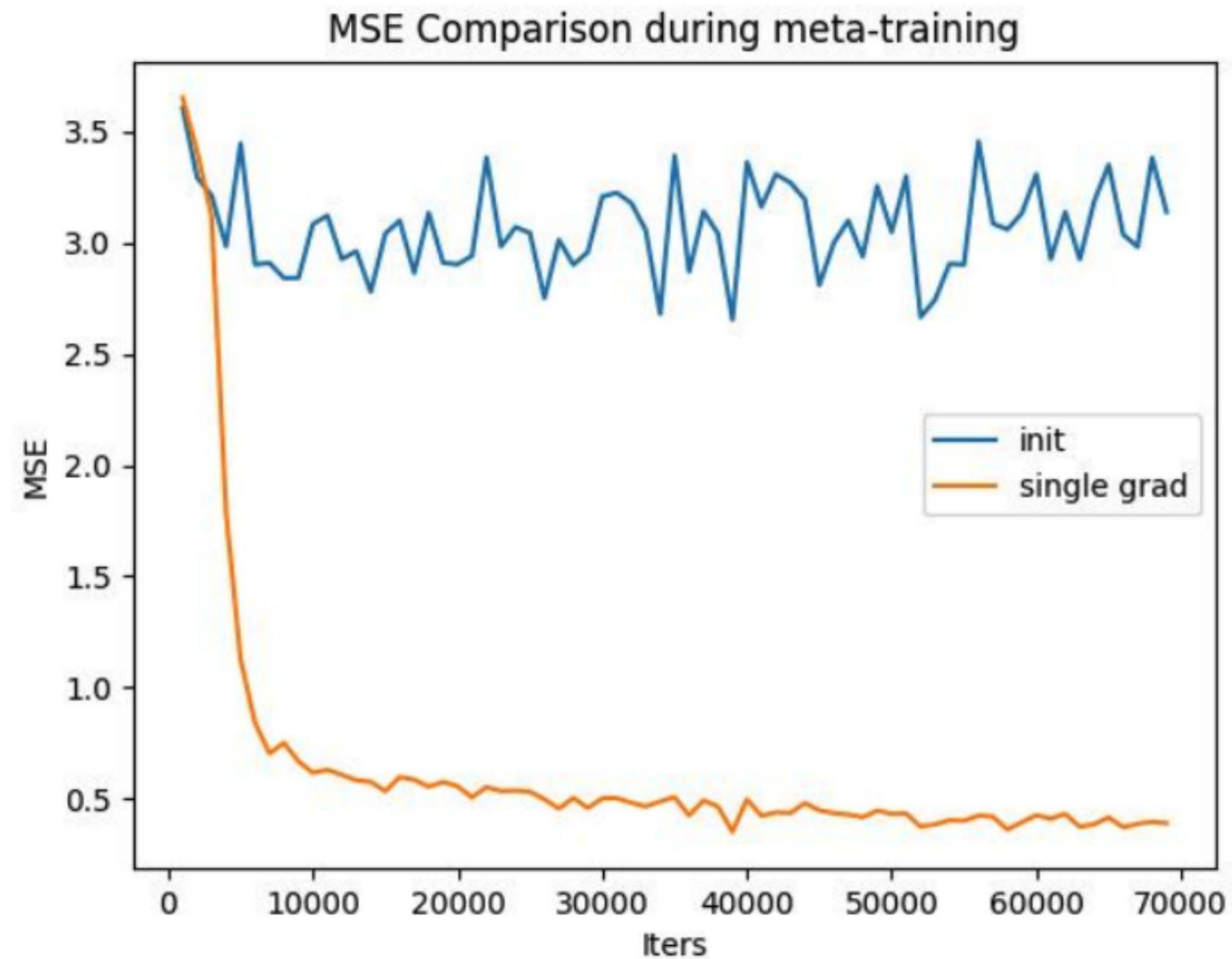


Vanilla pretrained model adapted slowly,  
but, the MAML method quickly adapted **even in one gradient step**.

Meta parameter theta of MAML is **more sensitive**  
than pre-updated theta of vanilla pretrained model.

## 5. Experiment

# MSE of the Meta-Parameters



- The performance of the meta parameters was not improved much in training.
- However, the performance of the single gradient updated parameters started on meta-parameters improved as training progressed.

## 5. Experiment

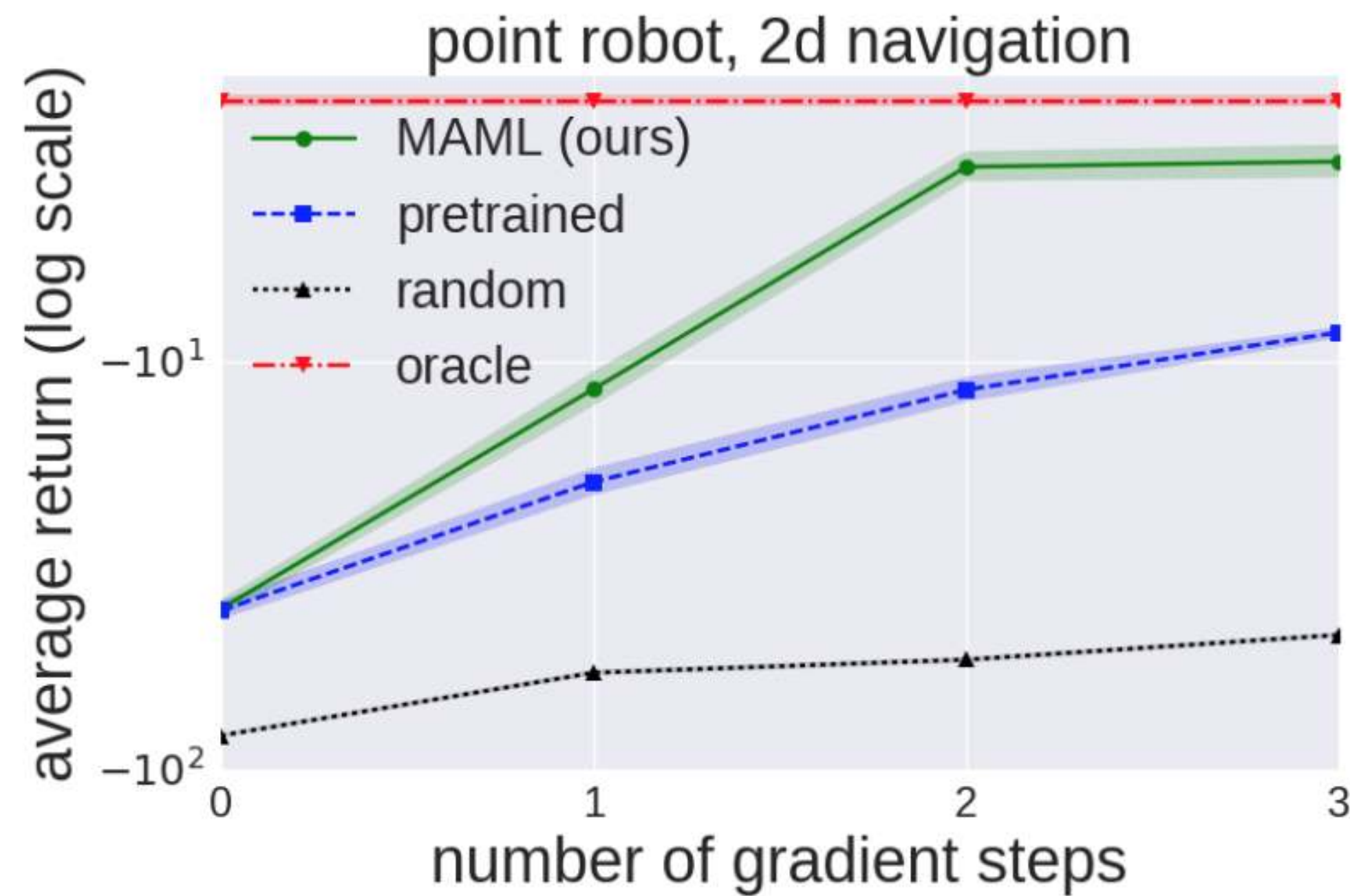
# Experiments on Reinforcement Learning

- rllab benchmark suite
- Neural network policy with two hidden layers of size 100 with ReLU
- Gradients updates are computed using vanilla policy gradient (REINFORCE) and trust-region policy (TRPO) optimization as meta-optimizer.
- Comparison
  - Pretraining one policy on all of the tasks and fine-tuning
  - Training a policy from randomly initialized weights
  - Oracle policy

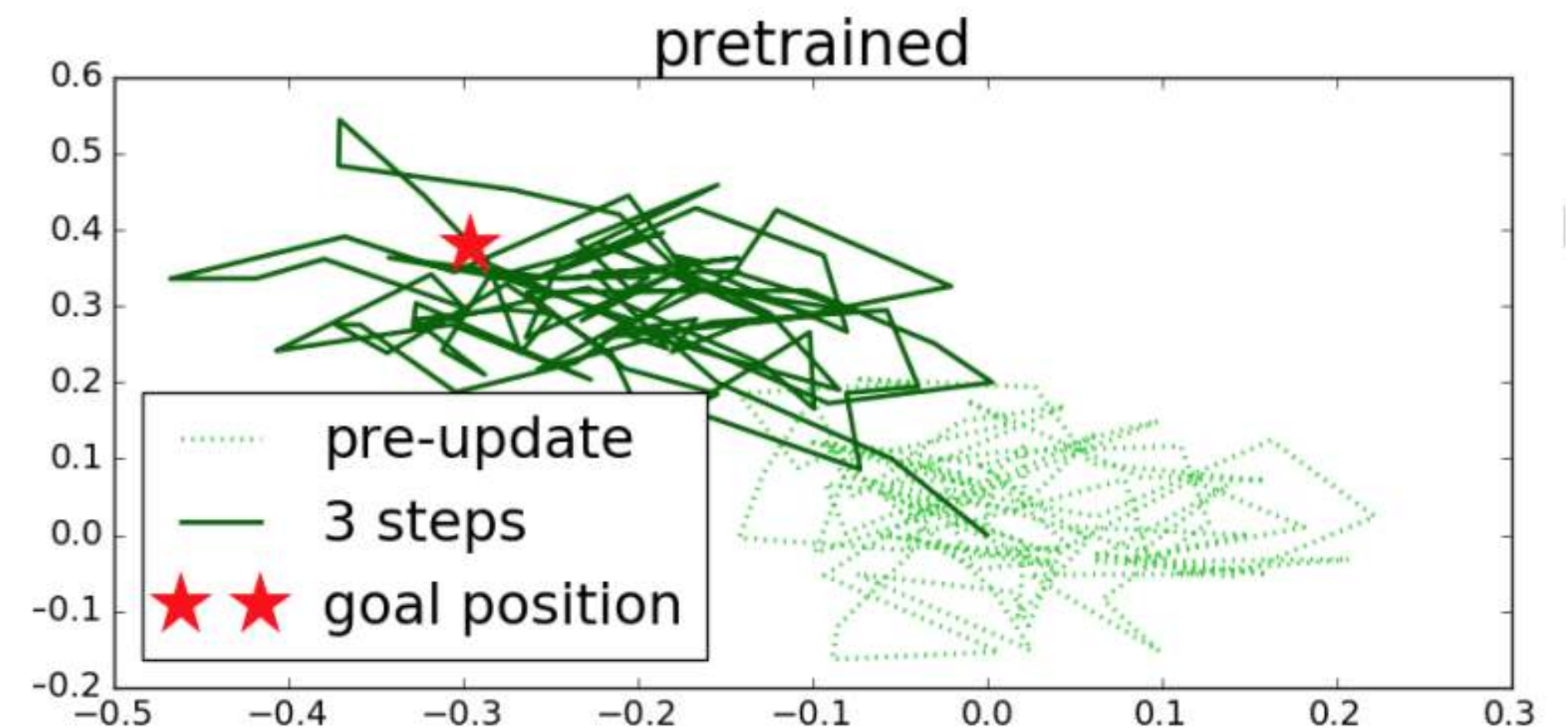
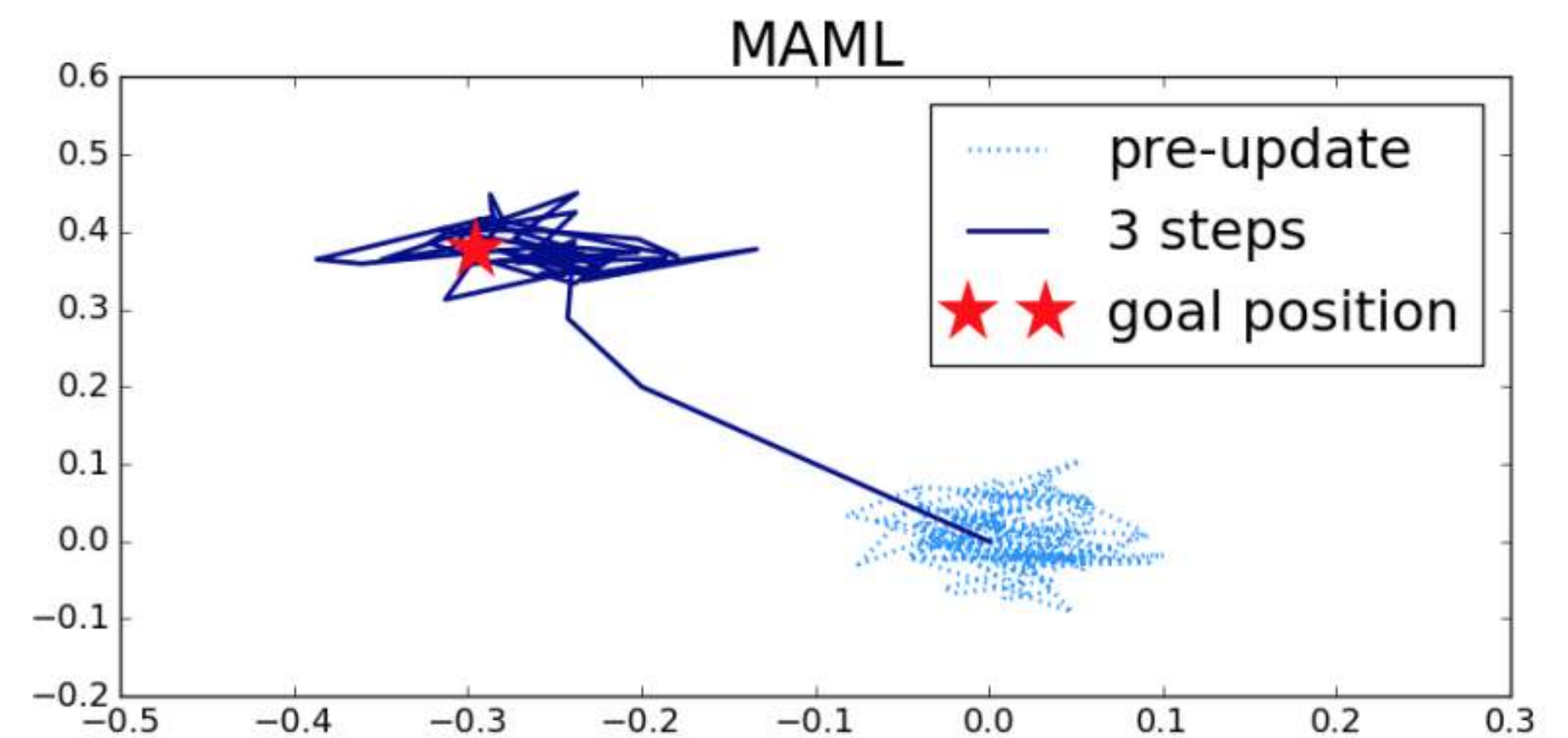
## 5. Experiment

# Results on Reinforcement Learning

- 2d navigation



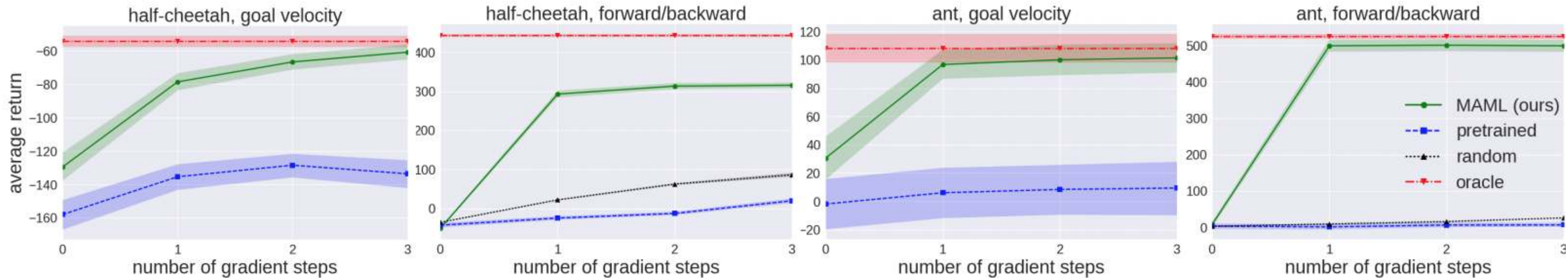
num. grad steps	0	1	2	3
context vector	-42.42	-13.90	-5.17	<b>-3.18</b>
MAML (ours)	-40.41	<b>-11.68</b>	<b>-3.33</b>	-3.23



## 5. Experiment

# Results on Reinforcement Learning

- Locomotion
  - High-dimensional locomotion tasks with the MuJoCo simulator



num. grad steps	0	1	2	3
context vector	-40.49	-44.08	-38.27	-42.50
MAML (ours)	-50.69	<b>293.19</b>	<b>313.48</b>	<b>315.65</b>

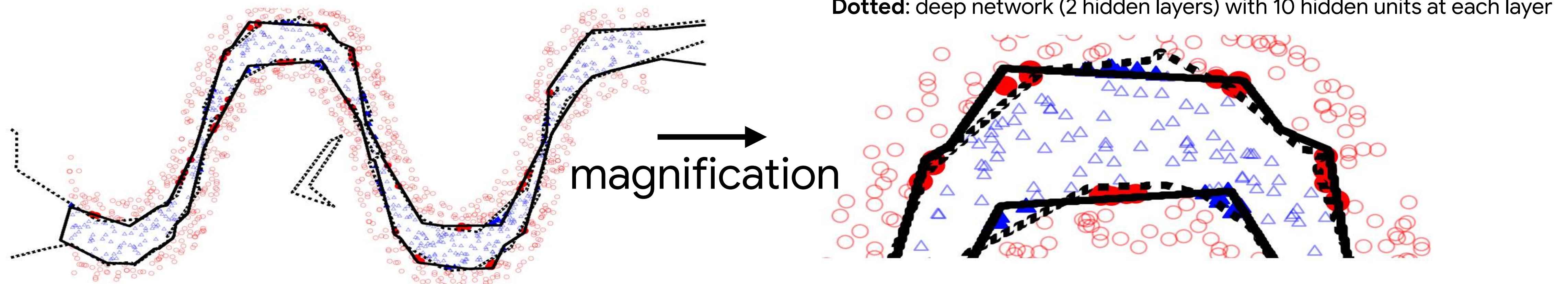
6. Discussion

# Why 1st Order Approximation Works?

## 6. Discussion

# Why 1st Order Approximation Works?

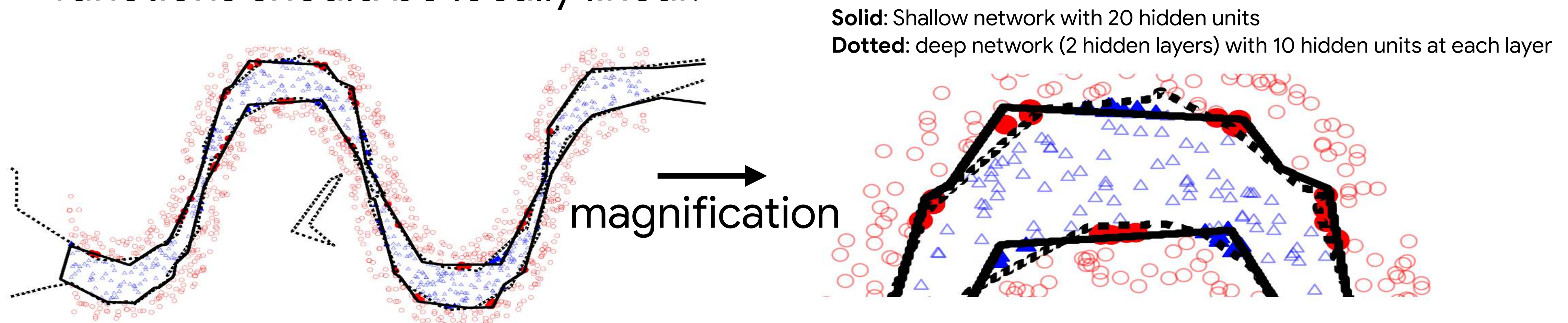
- [Montufar14] The objective function of ReLU network is locally linear.
- So as the other piecewise linear activations (maxout, leaky relu, ...) are used.
- Because, any composite function with linear or piecewise linear functions should be locally linear.



## 6. Discussion

# Why 1st Order Approximation Works?

- [Montufar14] The objective function of ReLU network is locally linear.
- So as the other piecewise linear activations (maxout, leaky relu, ...) are used.
- Because, any composite function with linear or piecewise linear functions should be locally linear.



- Hence, **the Hessian matrix is almost zero.**  
= Hessian matrix is meaningless.

## 6. Discussion

# Why 1st Order Approximation Works?

- If we change the activation function from ReLU to sigmoid?

## 6. Discussion

# Why 1st Order Approximation Works?

- If we change the activation function from ReLU to sigmoid?

	ReLU	sigmoid
w/o 1st order approx.	0.1183	0.1579
w 1st order approx.	0.1183	0.1931

[ MSE Comparison ]

	ReLU	sigmoid
w/o 1st order approx.	61	96
w 1st order approx.	53 (13% better)	52 (46% better)

[ Learning Time Comparison (minutes) ]

## 6. Discussion

# Why the Authors Focused on Deep Networks?

- The authors said their model (MAML) is model-agnostic.
- But, why they focused on deep networks?
- If it is not deep networks, what kind of problem can occur?

## 6. Discussion

# Why the Authors Focused on Deep Networks?

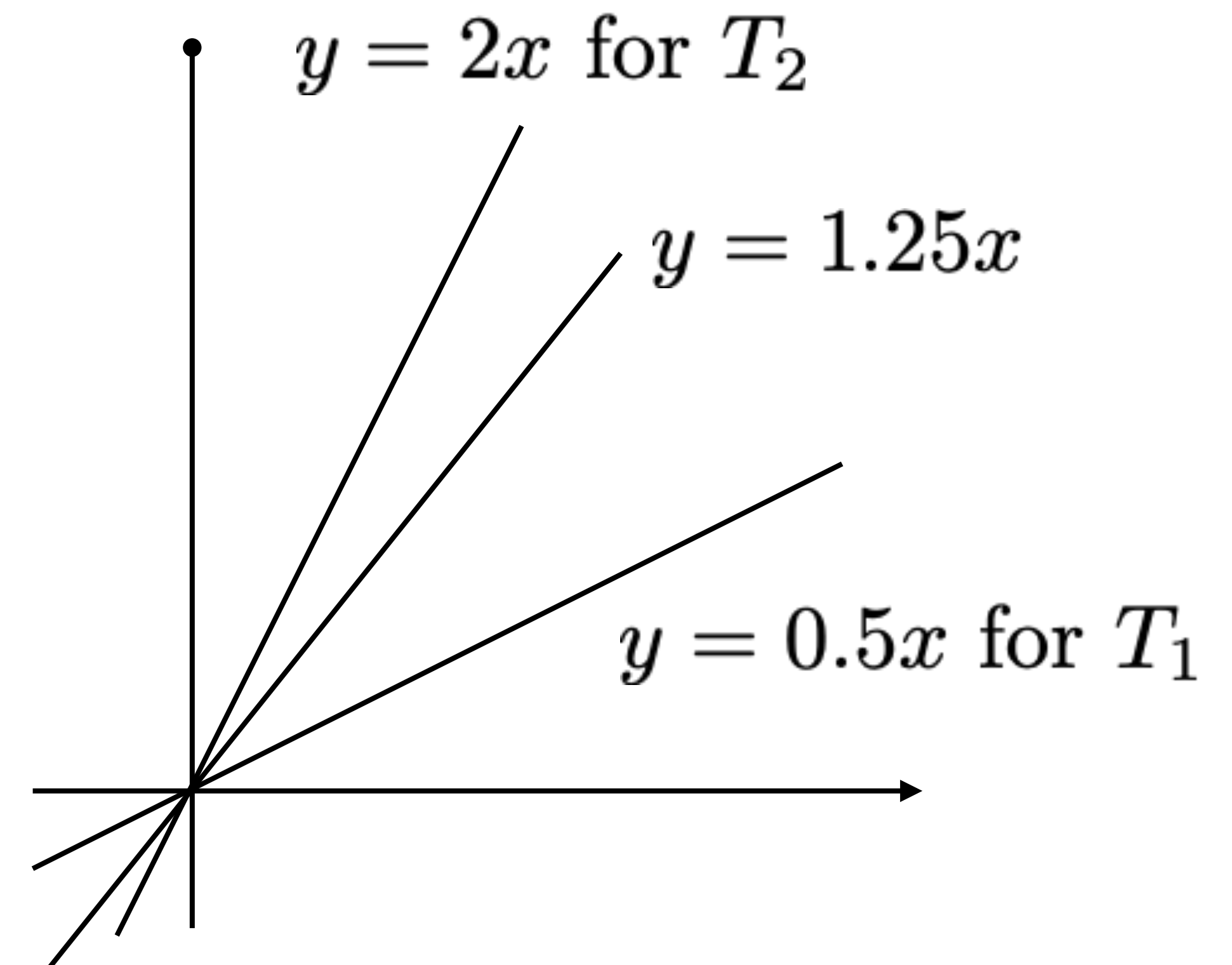
- Simple linear regression model ( $y = \theta x$ ) can be used in the MAML.
  - Parametrized, differentiable loss function.

## 6. Discussion

# Why the Authors Focused on Deep Networks?

- Simple linear regression model ( $y = \theta x$ ) can be used in the MAML.
  - Parametrized, differentiable loss function.

- Let  $T_i = \begin{cases} T_1 & \text{with probability 0.5.} \\ T_2 & \text{otherwise.} \end{cases}$   
 $y = 0.5x$  for  $T_1, y = 2x$  for  $T_2$ .



## 6. Discussion

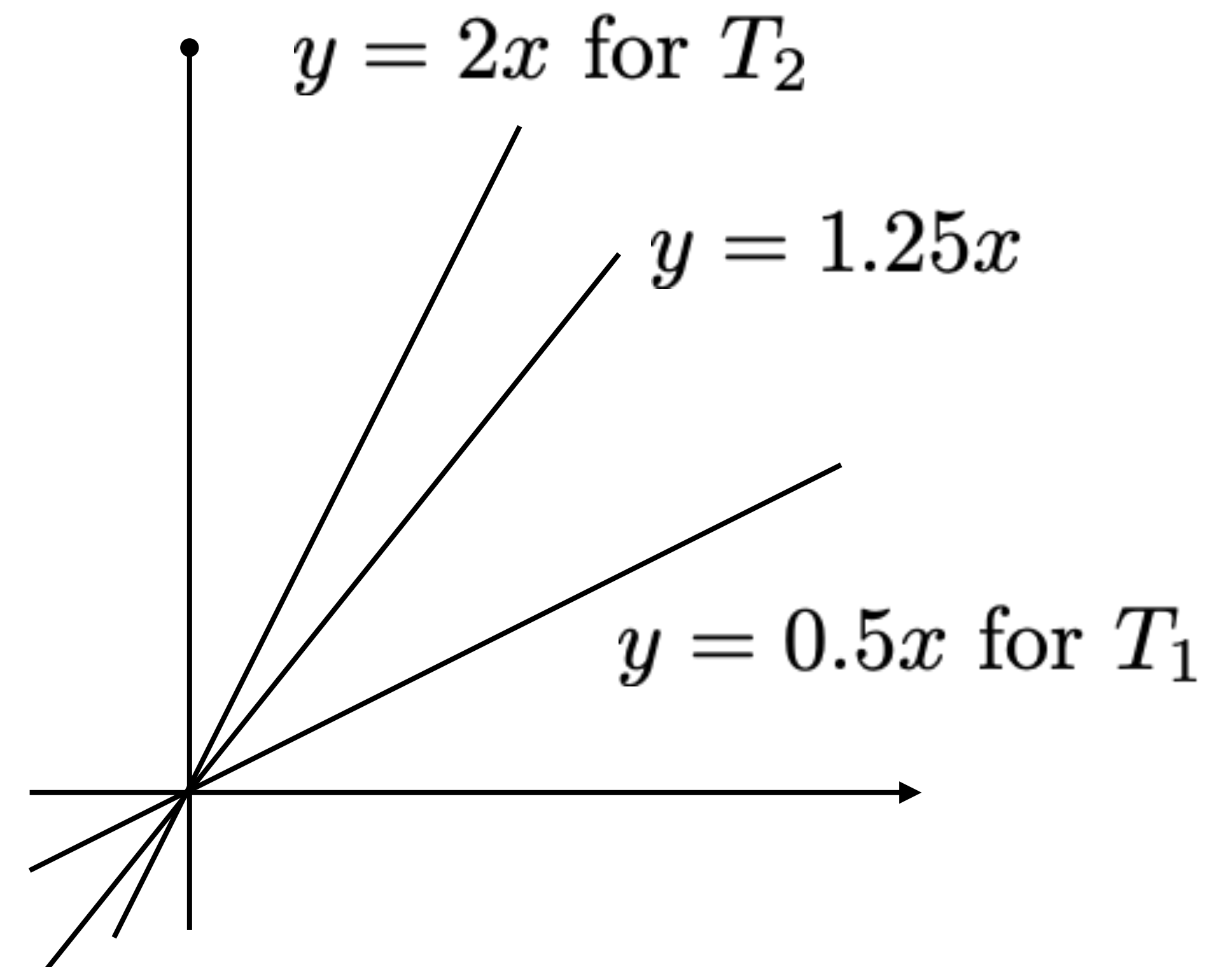
# Why the Authors Focused on Deep Networks?

- Simple linear regression model ( $y = \theta x$ ) can be used in the MAML.
  - Parametrized, differentiable loss function.

- Let  $T_i = \begin{cases} T_1 & \text{with probability 0.5.} \\ T_2 & \text{otherwise.} \end{cases}$

$$y = 0.5x \text{ for } T_1, y = 2x \text{ for } T_2.$$

- Then, meta parameter  $\theta$  will be 1.25.



## 6. Discussion

# Why the Authors Focused on Deep Networks?

- Let's extend the setting.
  - The sample is drawn from  $y = \theta_i x$  for the task  $T_i$ .
  - Then the meta parameter  $\theta = \mathbb{E}_{i \sim p(i)}[\theta_i]$ .

## 6. Discussion

# Why the Authors Focused on Deep Networks?

- Let's extend the setting.
  - The sample is drawn from  $y = \theta_i x$  for the task  $T_i$ .
  - Then the meta parameter  $\theta = \mathbb{E}_{i \sim p(i)}[\theta_i]$ .
- The MAML use fixed learning rate.
- More than 2 gradient steps are needed when  $\theta_i$  is far from  $\theta$ .

## 6. Discussion

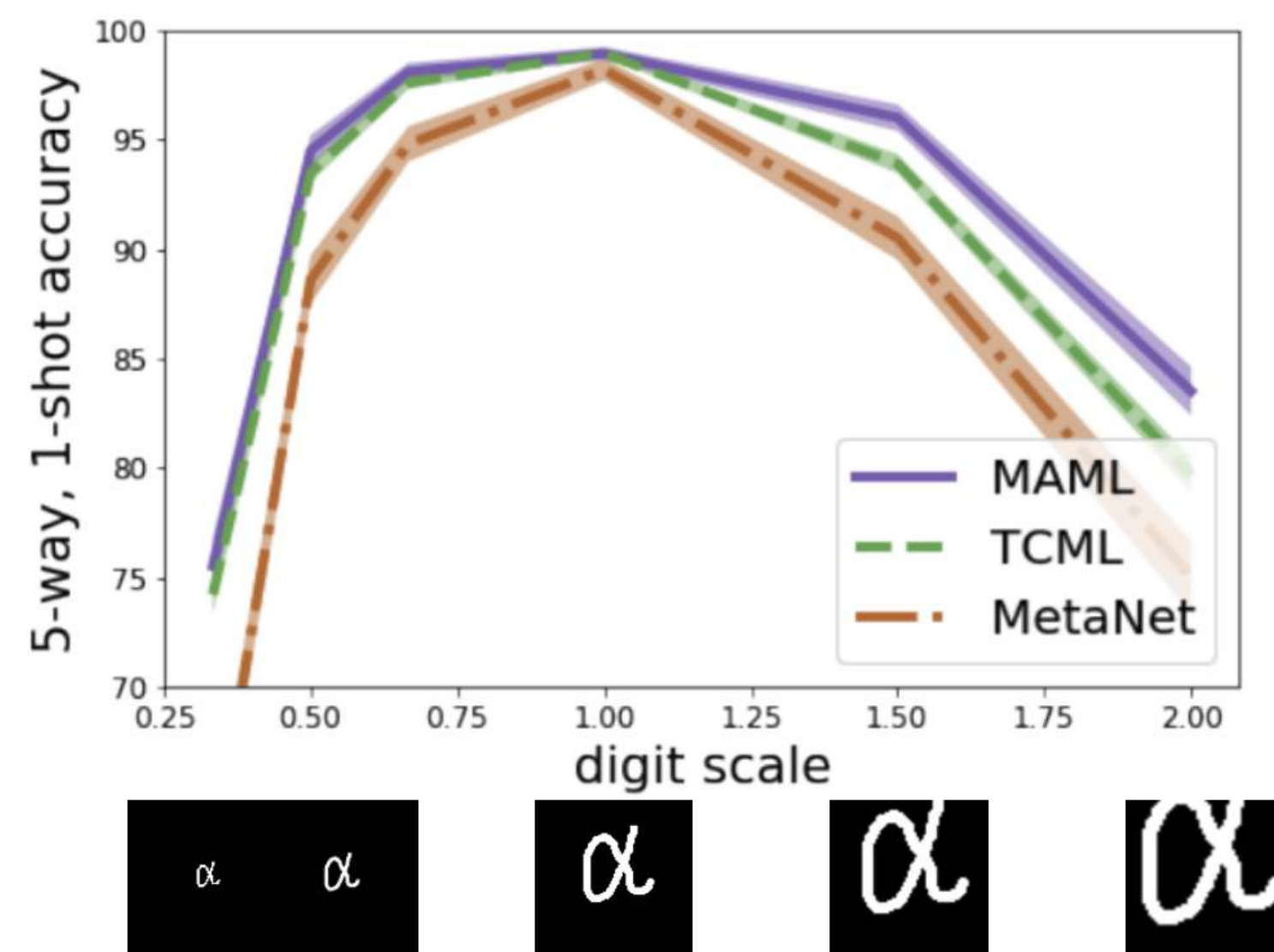
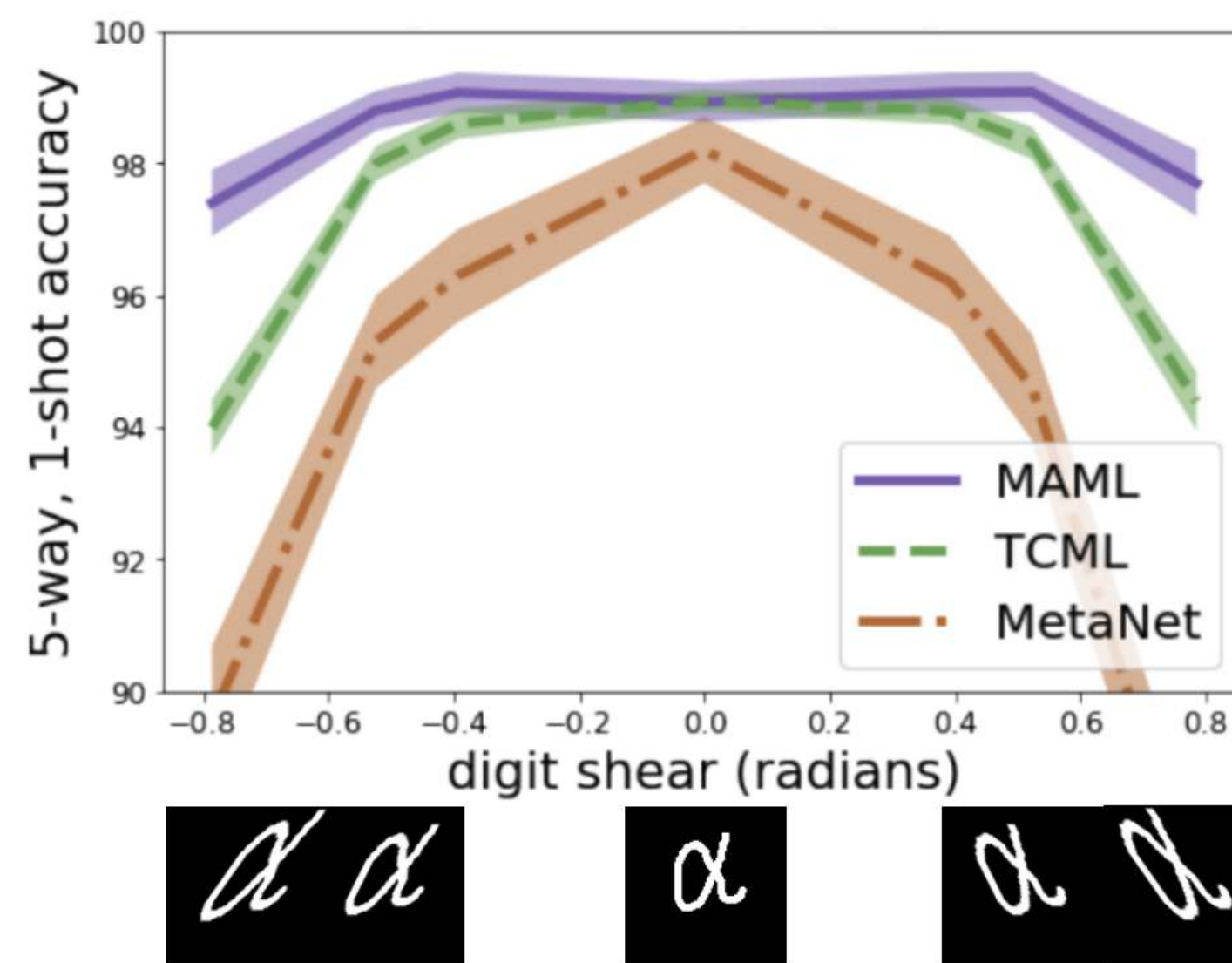
# Why the Authors Focused on Deep Networks?

- Let's extend the setting.
  - The sample is drawn from  $y = \theta_i x$  for the task  $T_i$ .
  - Then the meta parameter  $\theta = \mathbb{E}_{i \sim p(i)}[\theta_i]$ .
- The MAML use fixed learning rate.
- More than 2 gradient steps are needed when  $\theta_i$  is far from  $\theta$ .
- However, because the dimension of the deep model is large, there may be a meta parameter  $\theta$  that can quickly adapt to any task parameter  $\theta_i$ .
- This is why we think the authors focused on the deep models.

## 6. Discussion

# Why the Authors Focused on Deep Networks?

- However, using the deep model doesn't eliminate the problem completely.
- The authors mentioned about the generalization problem for extrapolated tasks.
- The closer the task is to out-of-distribution, the worse the performance.



- How can we solve this problem?

## 6. Discussion

# Update Meta Parameters with N-steps

- The MAML updates using only one gradient step when updating the meta parameter.
- The authors say that you can update meta parameters using n gradient steps.

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}) \longrightarrow \begin{aligned} \theta_i^1 &= \theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta}) \\ \theta_i^2 &= \theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i^1}) \\ &\dots \\ \theta_i^n &= \theta - \alpha \nabla \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i^{n-1}}) \\ \theta'_i &= \theta_i^n \end{aligned}$$

## 6. Discussion

# Update Meta Parameters with N-steps

- Will it show better performance to update with N gradient steps?
  - If so, why the authors didn't use this method in experiments?

## 6. Discussion

# Update Meta Parameters with N-steps

- Will it show better performance to update with N gradient steps?
  - If so, why the authors didn't use this method in experiments?
- If we apply this method to MAML, the MAML model will find a sensitive meta parameter in the N-step update.

## 6. Discussion

# Update Meta Parameters with N-steps

- Will it show better performance to update with N gradient steps?
  - If so, why the authors didn't use this method in experiments?
- If we apply this method to MAML, the MAML model will find a sensitive meta parameter in the N-step update.
- Also, we should calculate  $(n + 1)$ -order partial derivatives.
  - (c.f.) Hessian matrix is 2-order partial derivatives.
- The computation of  $(n + 1)$ -order partial derivatives is **computationally expensive**.

## 6. Discussion

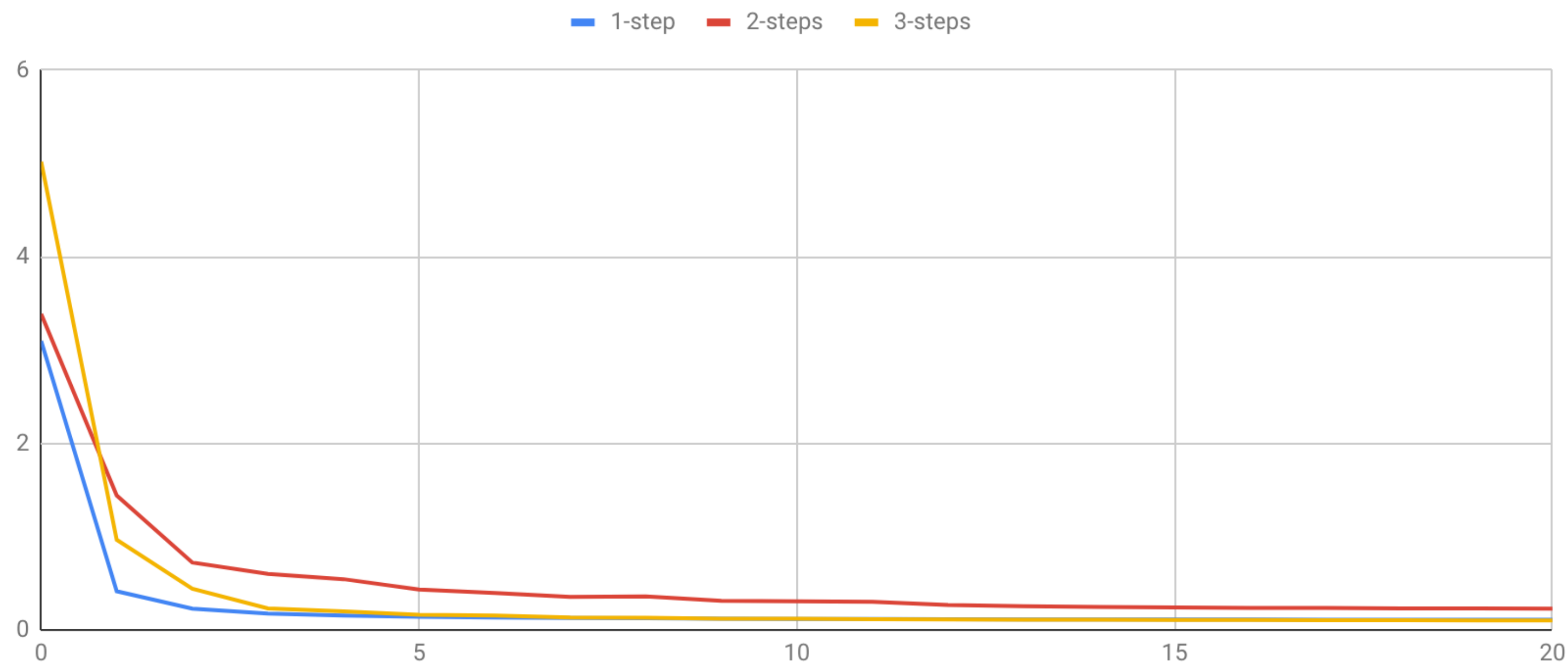
# Update Meta Parameters with N-steps

- How about using 1st order approximation with ReLU activation and updating meta parameter with N-steps?

## 6. Discussion

# Update Meta Parameters with N-steps

- N-step(s) updates with 1st order approximation.
- Of course, we use ReLU for activation function.

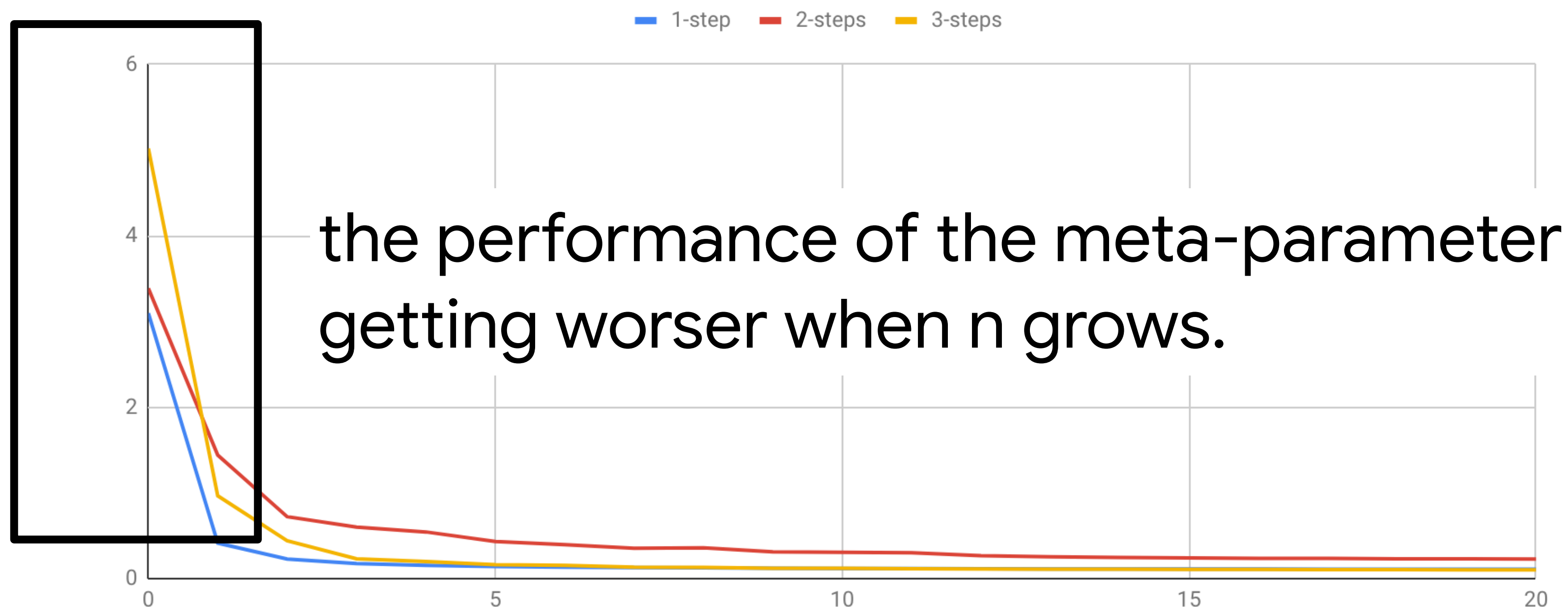


	1-step	2-steps	3-steps
Learning Time	53	73	92

## 6. Discussion

# Update Meta Parameters with N-steps

- N-step(s) updates with 1st order approximation.
  - Of course, we use ReLU for activation function.



	1-step	2-steps	3-steps
Learning Time	53	73	92

## 6. Discussion

# Update Meta Parameters with N-steps

- N-step(s) updates with 1st order approximation.
- Of course, we use ReLU for activation function.



	1-step	2-steps	3-steps
Learning Time	53	73	92

## 6. Discussion

# Unsupervised Learning

- The authors said that the MAML is model agnostic.
- They showed the MAML is worked on supervised learning.
- How about unsupervised learning?

## 6. Discussion

# Unsupervised Learning

- The authors said that the MAML is model agnostic.
  - They showed the MAML is worked on supervised learning.
  - How about unsupervised learning?
- 
- We can use autoencoder.
  - Autoencoder can make unsupervised learning be regarded as supervised learning.

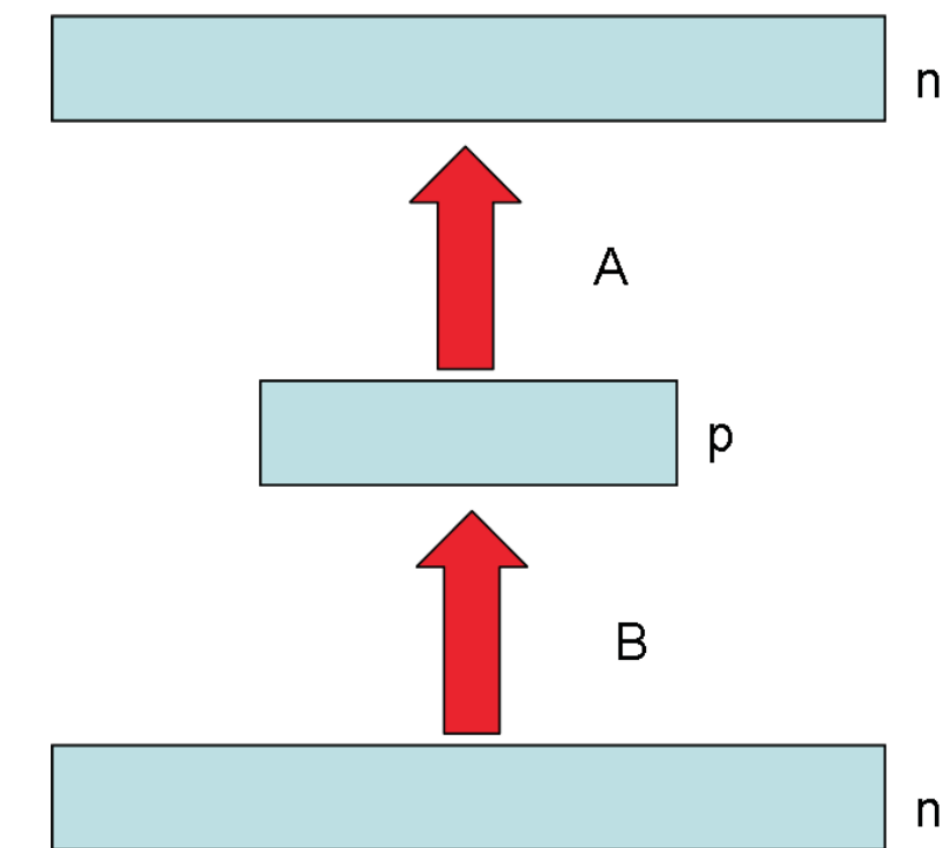


Figure 1: An  $n/p/n$  Autoencoder Architecture.

## 6. Discussion

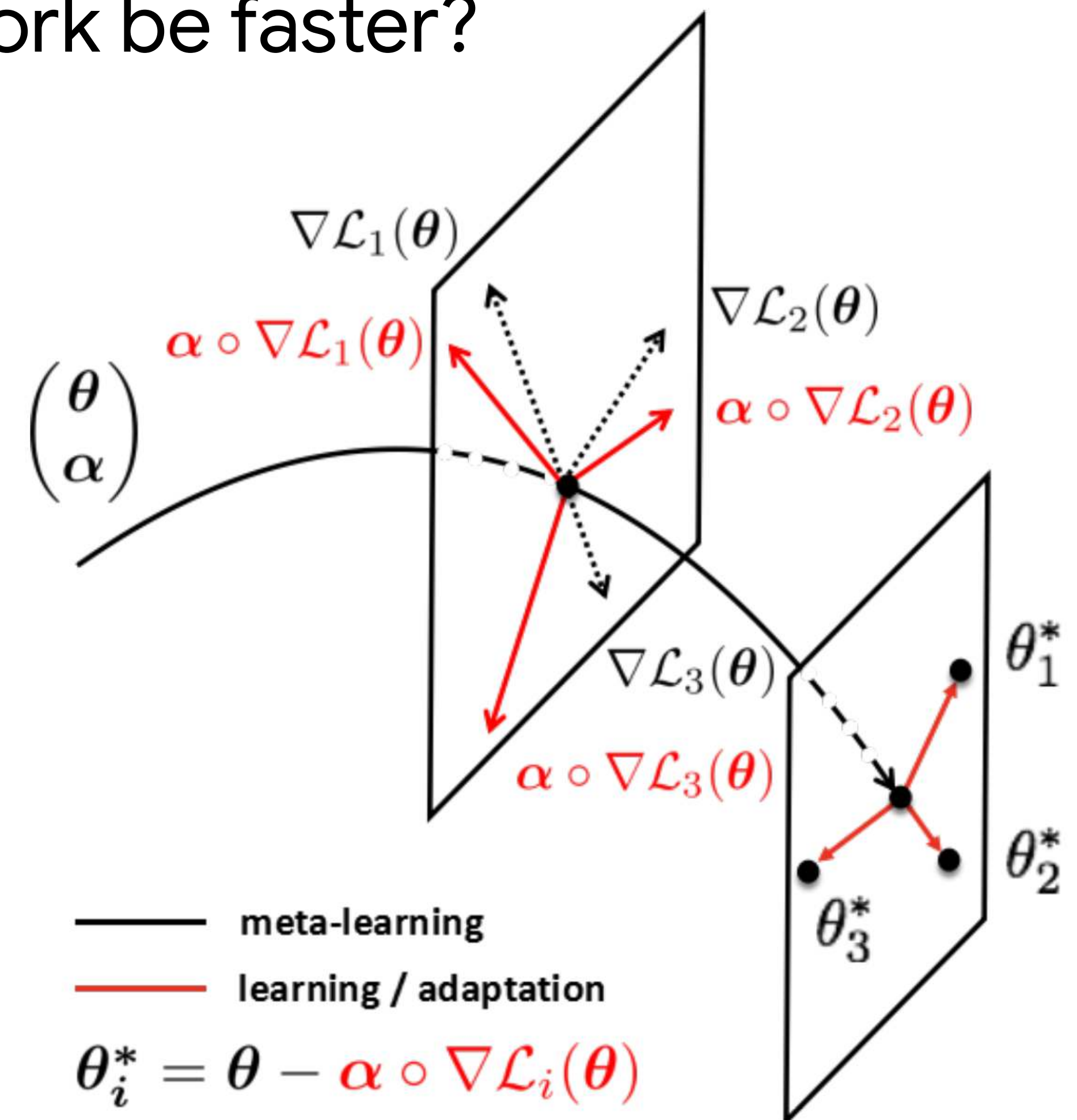
# How to Make the MAML be faster?

- MAML is really slow algorithm even with 1st order approximation.
- How can we make the MAML framework be faster?

## 6. Discussion

# How to Make the MAML be faster?

- MAML is really slow algorithm even with 1st order approximation.
- How can we make the MAML framework be faster?
- [Zhenguo 2017] suggests optimizing learning rate alpha.
- Instead of putting alpha as a hyperparameter, train **the vector** alpha as the parameters with other parameters.
- It outperforms the vanilla MAML.



## 6. Discussion

# Application

- The MAML is good for a few-shot learning problem.
- Can you guess any applications of few-shot learning problem?

## 6. Discussion

# Application

- The MAML is good for a few-shot learning problem.
- Can you guess any applications of few-shot learning problem?
- We think one of the most effective few-shot learning problem is real estate price prediction.
  - There are a lot of transactions related to big apartments.
  - But, there are many transactions in small apartments.  
So, it is difficult to predict.
  - Using MAML seems to solve the problem of insufficient sample size of small apartments.
- Also, other problems would be solved by MAML.

Thank you!

# Reference

- Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." arXiv preprint arXiv:1703.03400 (2017).
- Montufar, Guido F., et al. "On the number of linear regions of deep neural networks." Advances in neural information processing systems. 2014.
- Baldi, Pierre. "Autoencoders, unsupervised learning, and deep architectures." Proceedings of ICML workshop on unsupervised and transfer learning. 2012.
- Li, Zhenguo, et al. "Meta-sgd: Learning to learn quickly for few shot learning." arXiv preprint arXiv:1707.09835 (2017)

# Reference

- <https://www.slideshare.net/TaesuKim3/pr12094-modelagnostic-metalearning-for-fast-adaptation-of-deep-networks>
- [https://people.eecs.berkeley.edu/~cbfinn/\\_files/nips2017\\_metaworkshop.pdf](https://people.eecs.berkeley.edu/~cbfinn/_files/nips2017_metaworkshop.pdf)
- <https://github.com/cbfinn/maml>
- Lab seminar slide by Young-ki Hong.