Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Chelsea Finn, Pieter Abbeel, Sergey Levine

University of California, Berkeley and OpenAI

ICML 2017

Versatility of intelligence

- The capability of doing many different things
- Current AI systems excel at mastering a single skill
- But struggle in doing a variety of simple problems
  - A champion can not hold a conversation
  - A expert helicopter controller for aerobatics can not navigate in new, simple situations

Meta-learning: Learning to learn

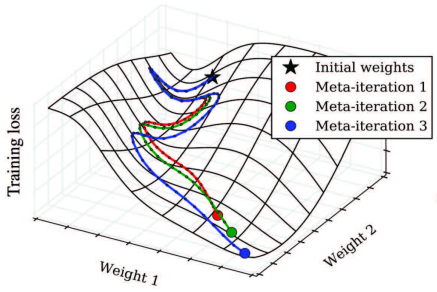
- Current AI systems
  - Master a complex skill from scratch
  - Using an understandably large amount of time and experience
- Acquire many skills and adapt to many environments
  - Cannot afford to train each skill in each setting from scratch.
- Learn how to learn new tasks faster by reusing previous experience
  - Rather than considering each new task in isolation

Why learning to learn

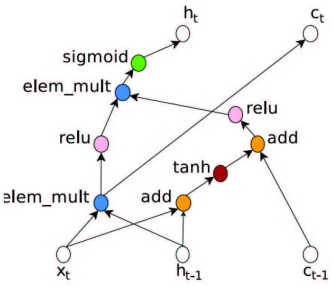
- Effectively reuse data on other tasks
- Replace manual engineering of architecture, hyperparameters, etc.
- Learn to quickly adapt to unexpected scenarios
  - Inevitable failures
  - Long tail
- Learn how to learn with weak supervision

The use of learning to learn

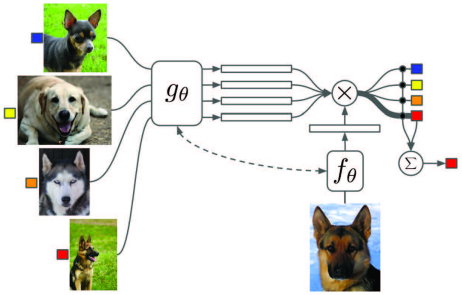
- Hyperparameter and neural network optimization
- Finding good network architectures
- Few-shot image recognition
- Learning to efficiently navigate a new maze with only one traversal through the maze



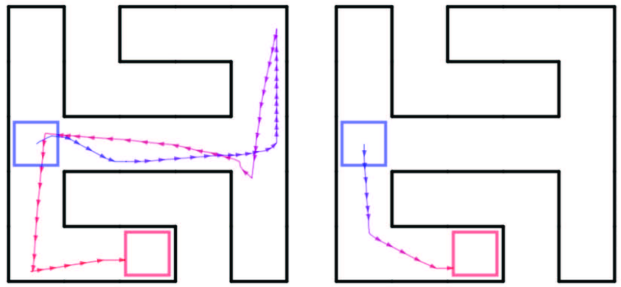
hyperparameter optimization
Maclaurin et al. '15



learned recurrent cell
Zoph & Le '17



few-shot image classifier
Vinyals et al. '16

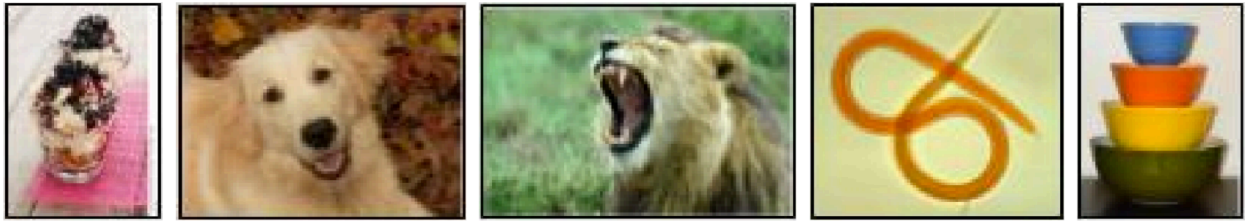


learning to quickly navigate new mazes
Duan et al. '16

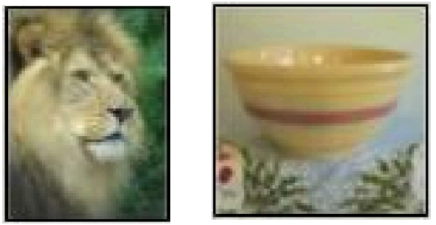
Typical Objective of Few-Shot Learning

Image recognition

Given 1 example of 5 classes:

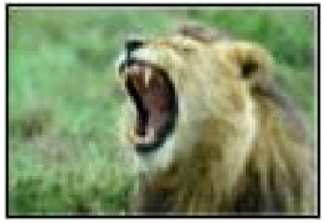


Classify new examples



Human Concept Learning

Given *1 positive example*:



Classify new examples:

Beyond how humans learn, this setting is also more interesting.

Recent Meta-learning Approaches

- Standard ML:
  - Training on a single task and testing on held-out examples from that task
- Meta-learning:
  - Training with a large number of tasks
  - Testing in their ability to learn new tasks
- Two optimizations at the same time
  - The learner: learn new tasks
  - The meta-learner: train the learner

The Meta-Learning Problem

Supervised Learning:

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

$$\mathbf{y} = f(\mathbf{x}; \theta)$$

Meta-Supervised Learning:

Inputs: $\mathcal{D}_{\text{train}}$ $\mathbf{x}_{\text{test}}$ Outputs: $\mathbf{y}_{\text{test}}$ Data: $\{\mathcal{D}_i\}$

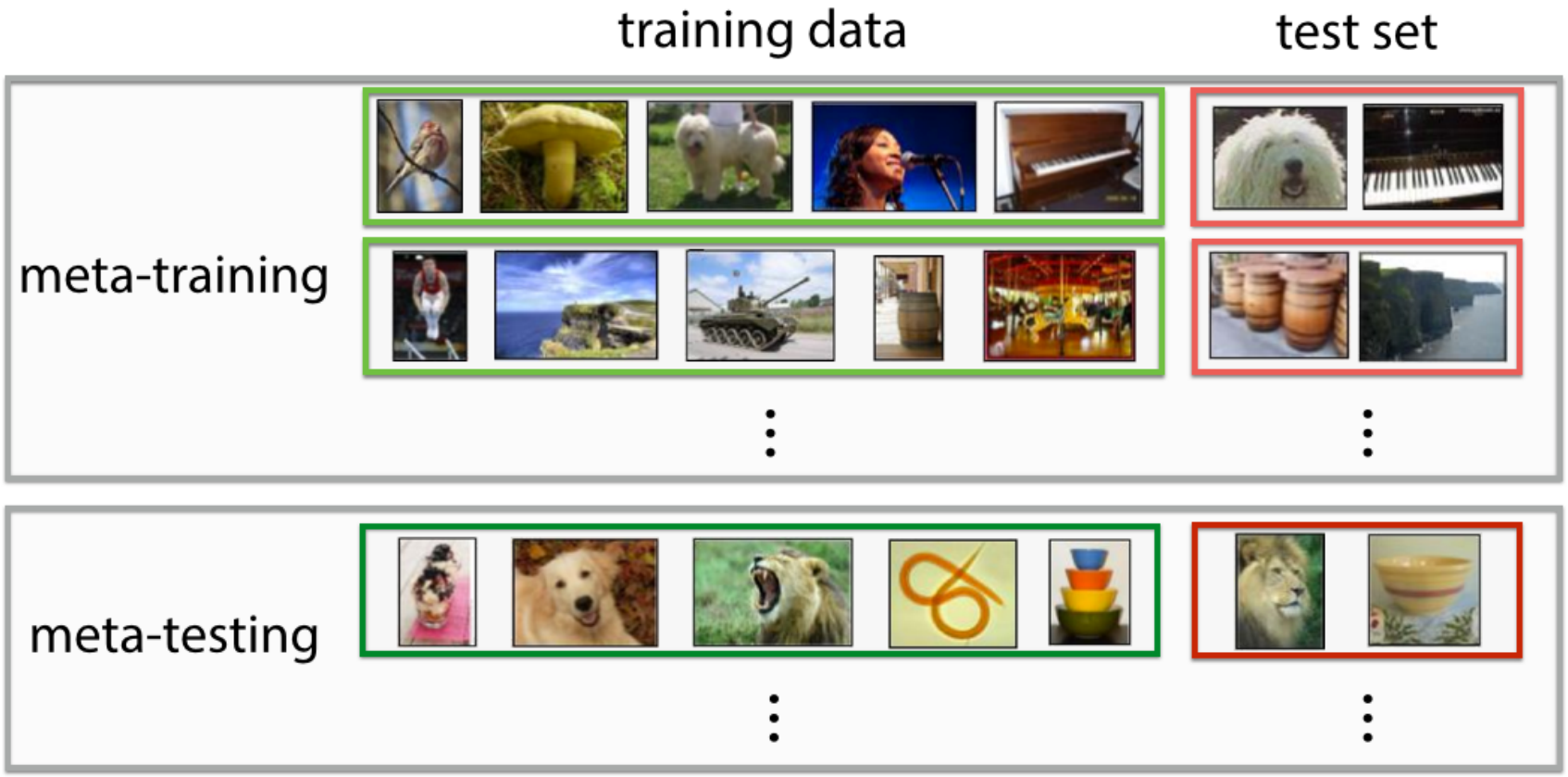
$\{(\mathbf{x}, \mathbf{y})_{1:K}\}$

$$\mathbf{y}_{\text{test}} = f(\mathcal{D}_{\text{train}}, \mathbf{x}_{\text{test}}; \theta)$$

$$\mathcal{D}_i : \{(\mathbf{x}, \mathbf{y})_j\}$$

Meta-learning Example

- Classifying a new image within 5 possible classes

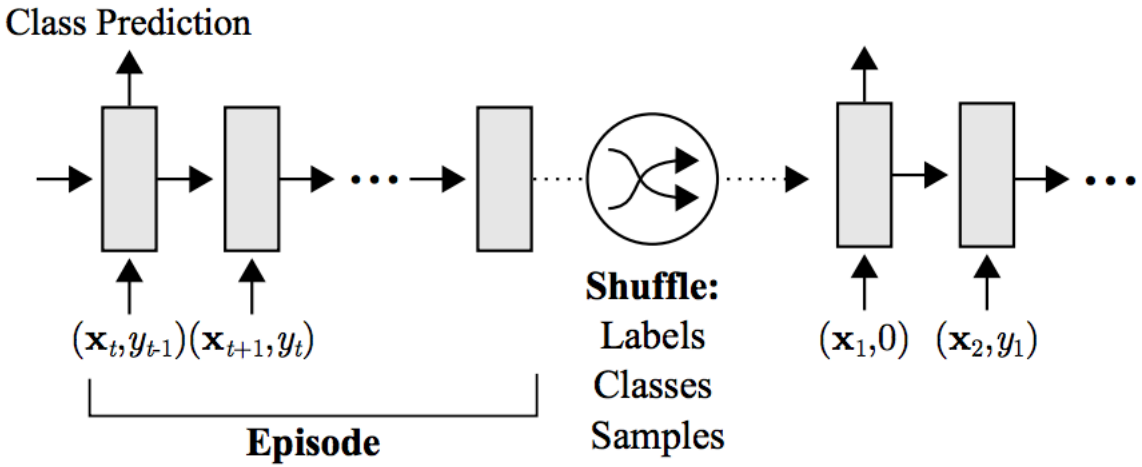


Recurrent Models

- Train a recurrent model
  - an LSTM
- Take in the dataset sequentially and then process new inputs from the task
- The recurrent learner is trained to adapt to new tasks as it is rolled out

Recurrent Models

- Image classification
  - Passing in the set of (image, label) pairs of a dataset sequentially
  - Followed by new examples which must be classified



Metric Learning

- Learn a metric in input space
  - A way of measuring the distance (or similarity) between objects
- Specialized to one/few-shot classification
- Can not be used in other problem
  - RL
- Meta-learner: gradient descent
- Learner: a comparison scheme
  - Nearest neighbors in the meta-learned metric space
  - Euclidean distance

Learning Optimizers

- Better neural network optimization
- Learn parameter update when given gradients
  - Search space includes SGD, RMSProp, Adam, etc.
- Meta-learner: a recurrent network
  - remember how it previously updated the learner model
  - Trained with reinforcement learning or supervised learning
- Few-shot image classification

Model-Agnostic Meta Learning

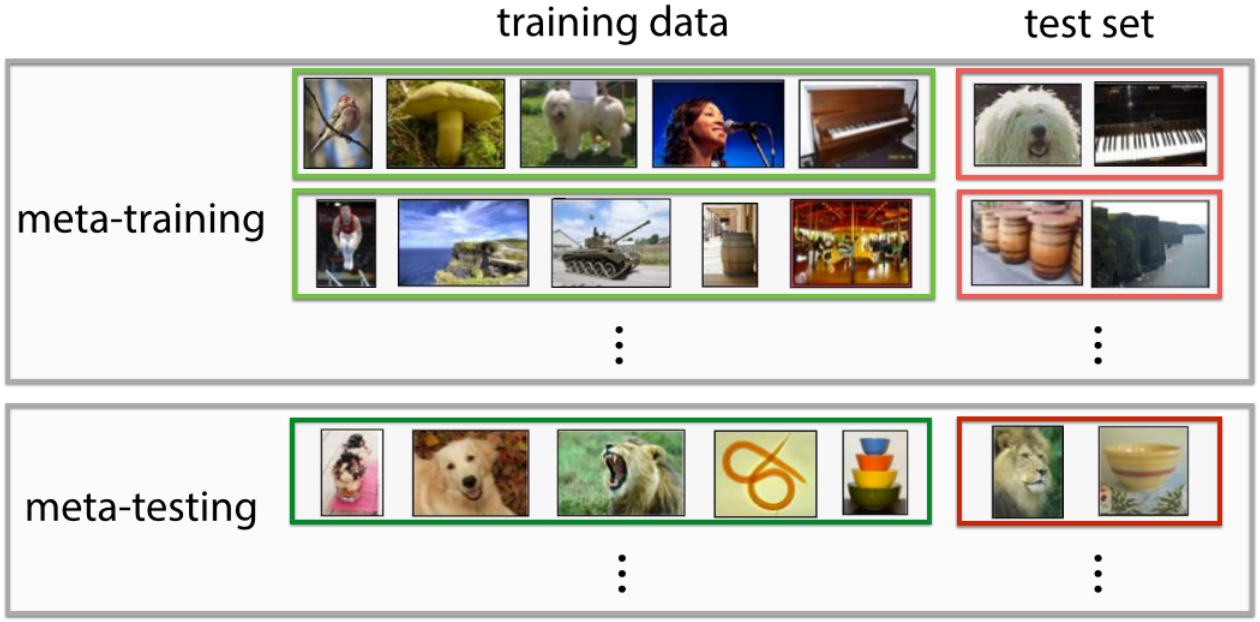
- Trained on a set of tasks
  - Adapt to new tasks quickly using only a small number of examples or trials
- Model f: map observations x to outputs a
- Learning task $\mathcal{T} = \{\mathcal{L}(x_1, a_1, \dots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$
  - A loss function $\mathcal{L}$: a mis-classification loss or a cost function in MDP
  - A distribution over initial observations $q(x_1)$
  - A transition distribution $q(x_{t+1}|x_t, a_t)$
  - An episode length H

MAML Training Phase

- Consider a distribution over tasks $p(\mathcal{T})$
- Loop for:
  - Learn a new task $\mathcal{T}_i$ drawn from $p(\mathcal{T})$ from only K samples and feedback
  - Test on new samples from $\mathcal{T}_i$
  - Consider how test error on new data changes w.r.t the parameters

Model-Agnostic Meta Learning

- Training error of meta learning process
  - Test error
- Meta-performance
  - The model's performance after learning from K samples

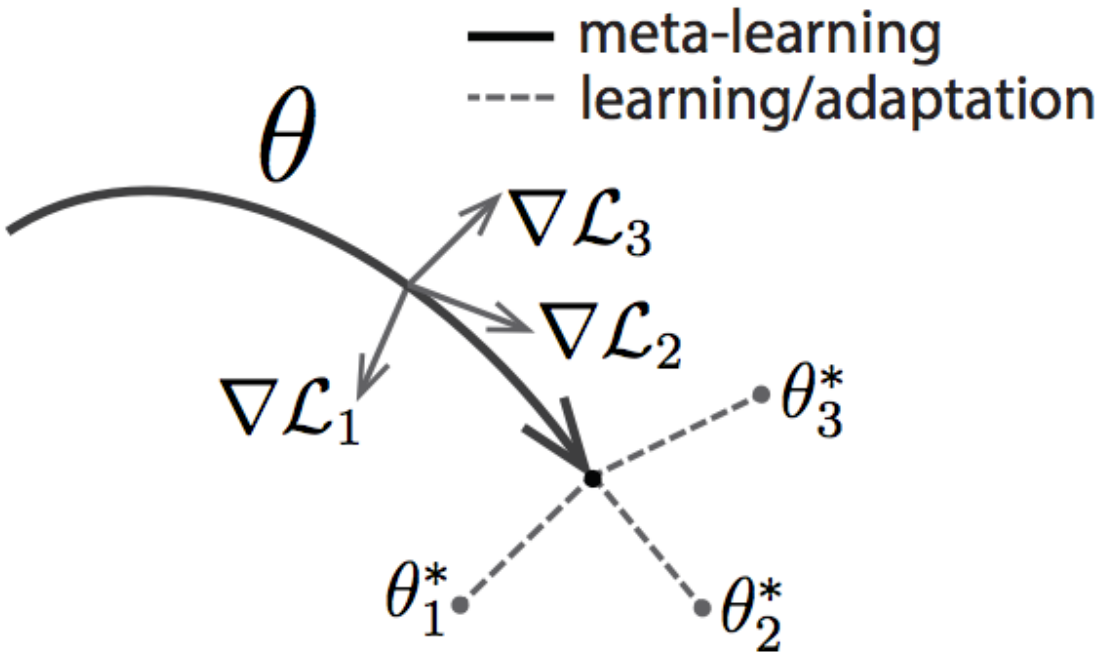


Algorithm

- Some internal representations are more transferable than others
- Learn internal features that are broadly applicable to all tasks in $p(\mathcal{T})$
  - Rather than a single individual task

Algorithm

- Find model parameters that are sensitive to changes in the task
  - Make rapid progress on new tasks
  - Small changes in the parameters will produce large improvements on the loss function of any task



fine-tuning:

[test-time]

pretrained parameters

$$\theta' \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

test task

Model-Agnostic Meta-Learning: (MAML)

$$\min_\theta \sum_{\text{tasks}} \mathcal{L}_{\text{v}}(\theta - \alpha \nabla_\theta \mathcal{L}_{\text{tr}}(\theta))$$

Key idea: Train over many tasks, to learn parameter vector θ that transfers

In-distribution task: k-shot learning

Base case: learning from scratch

Related but out-of-distribution task: somewhere in between

Algorithm

Algorithm 1 Model-Agnostic Meta-Learning

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

Require: α, β: step size hyperparameters

1: randomly initialize θ

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: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples

6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$

7: **end for**

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

9: **end while**

Sample different tasks

Update parameters of each task

Minimize the meta-objective: find parameter θ to minimize the loss of all tasks

Supervised Regression Classification

- Regression tasks: Use mean-squared error
- Discrete classification tasks: Use a cross entropy loss

Sample K data points using f_θ in $\mathcal{T}_i$

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
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: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Reinforcement Learning

- Query K sample trajectories for few-shot learning
  - Trajectory: a sequence of state-action pairs from starting state to ending state

Sample K trajectories using f_θ in $\mathcal{T}_i$

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
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: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Reinforcement Learning

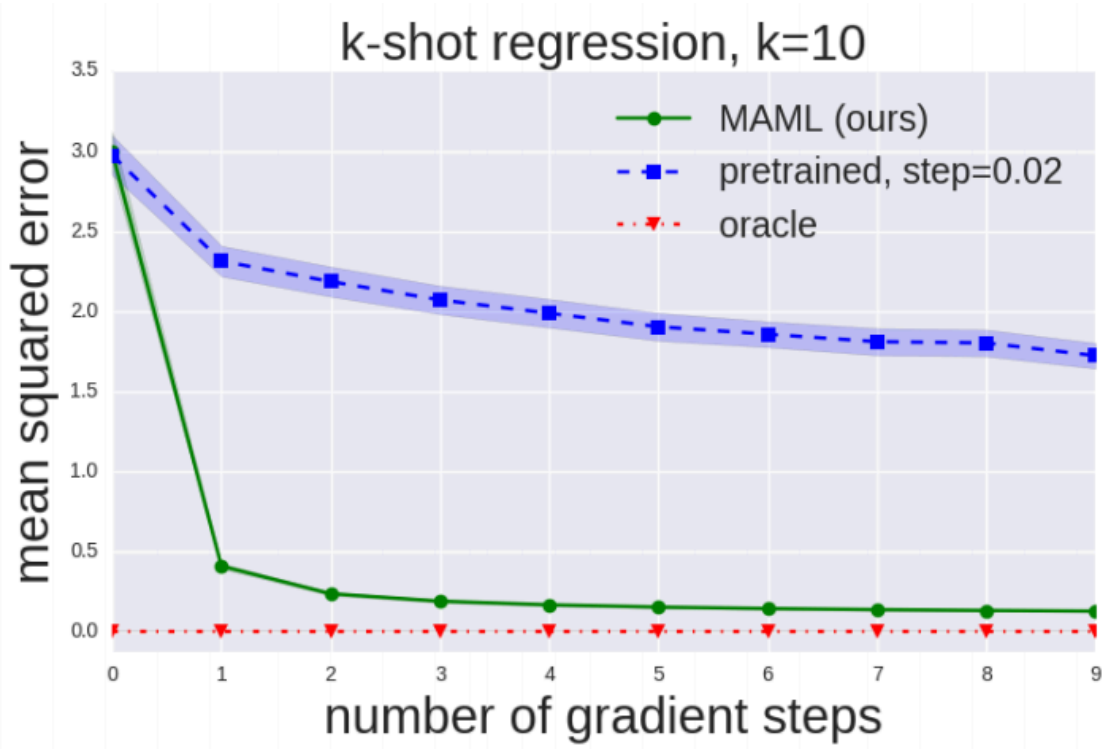
- Learn a policy
  - Map from states to a distribution over actions at at each time step
- Loss function
  - Sum of negative reward

Evaluation: Regression

- Each regression task maps input to the output of a sine wave
  - The amplitude and phase of the sinusoid are varied
  - $f_i(x) = A\sin(x - \phi)$
- Comparisons:
  - Pre-training on all tasks and fine-tuning on new tasks
    - Use SGD on the K provided points from new tasks
  - An oracle which receives the true A and ϕ as input

Evaluation: Regression

- In-distribution: $A \in [0, 5]$, $\phi \in [0, \pi]$
- Out-of-distribution: $A \in [5, 10]$, $\phi \in [\pi, 2\pi]$

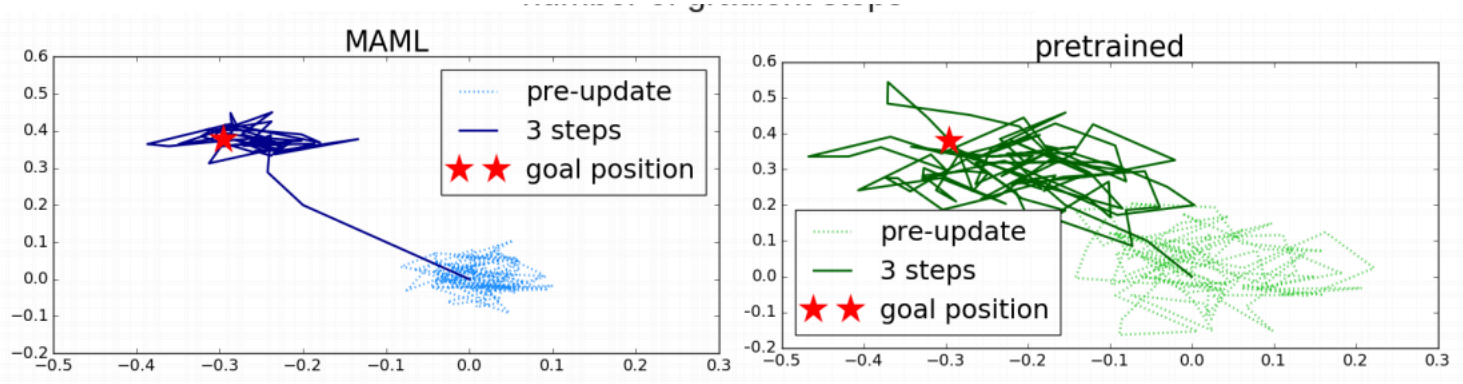


Evaluation: Reinforcement Learning

- A point agent must move to different goal positions in 2D square
  - Different tasks have different goal positions
  - Randomly chosen for each task within a unit square
- Observation
  - The current 2D position

Evaluation: Reinforcement Learning

- Actions
  - Velocity commands
  - With the range $[-0.1, 0.1]$
- Reward
  - The negative squared distance to the goal



Summary

- **Pros:**
- Reusing knowledge from past tasks
  - A crucial ingredient in making high-capacity scalable models
  - Lifelong few-shot learning
- Fast training with small datasets
- **Cons:**
- But needs to train on different types of tasks
- Multiple tasks are still related
  - The NN architecture needs to be the same
- The performance may not be as good as designing NN for a specific task