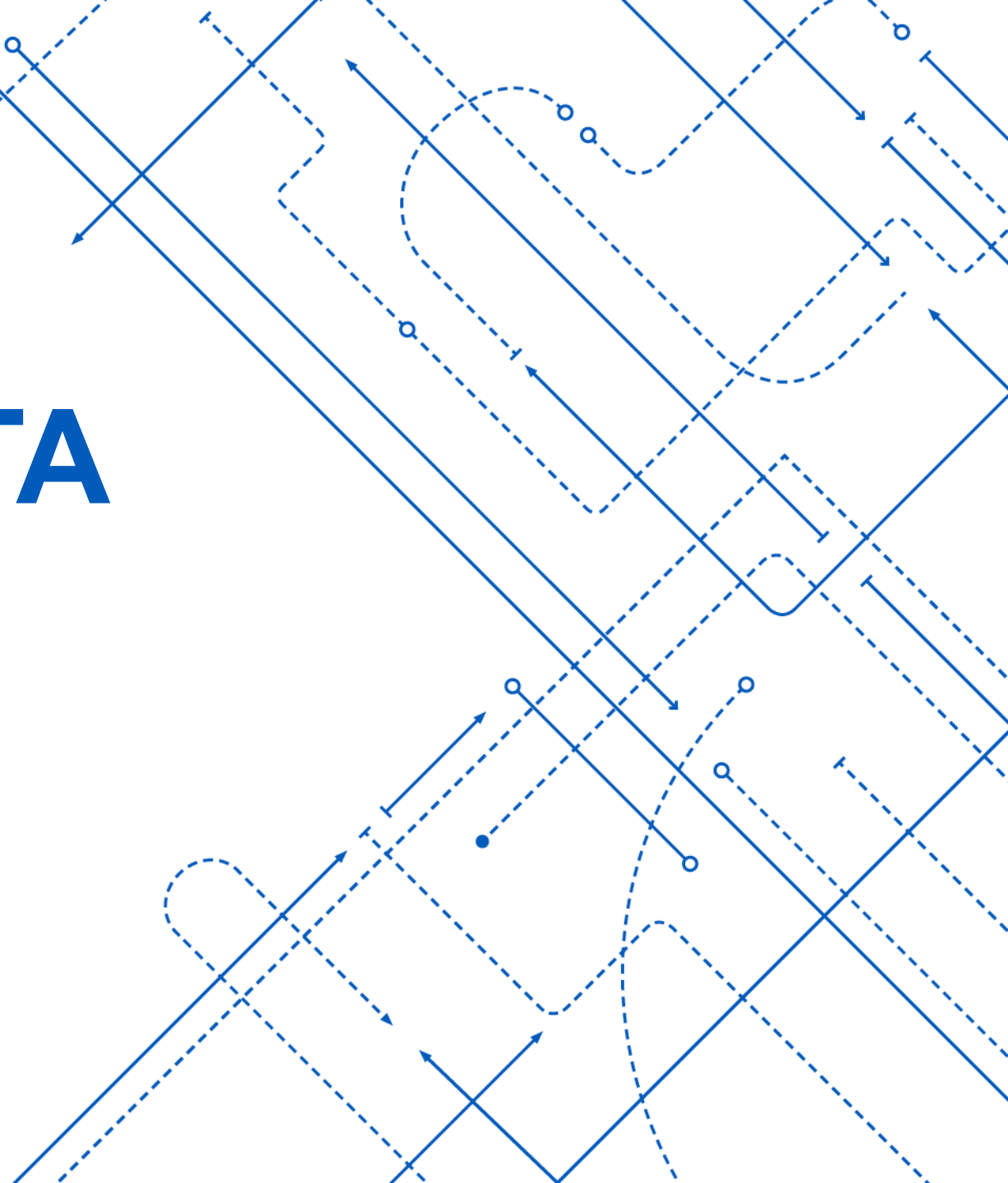
MODEL AGNOSTIC META LEARNING

Rohan Sharma

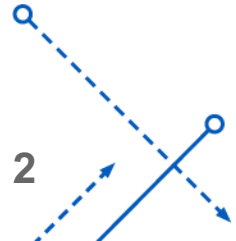
50431056

24-October-2022

University at Buffalo The State University of New York

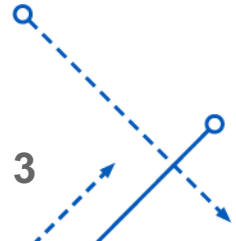


Contents



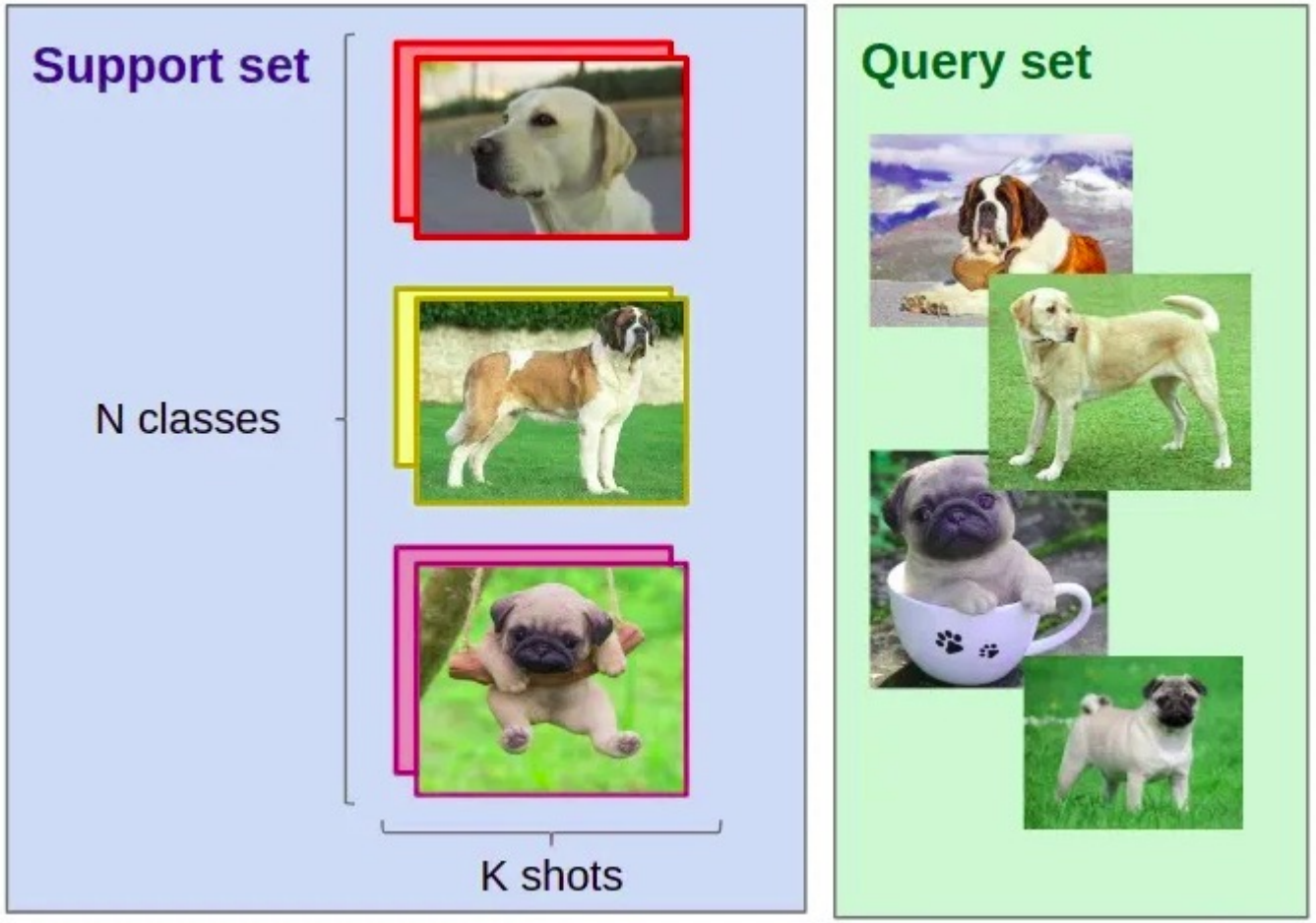
Introduction

- Meta learning essentially means learning to learn
- If a classifier learns to classify on a dataset with N classes, will it do well on N` new classes?
  - Why can’t we use the learned model again?
- Learn a model on a variety of learning tasks
- Now requires few-shot training on the new tasks with SOTA performance
- Easy to finetune
- Use the same model for both tasks and all others
- Generic learning algorithm that learns to learn!



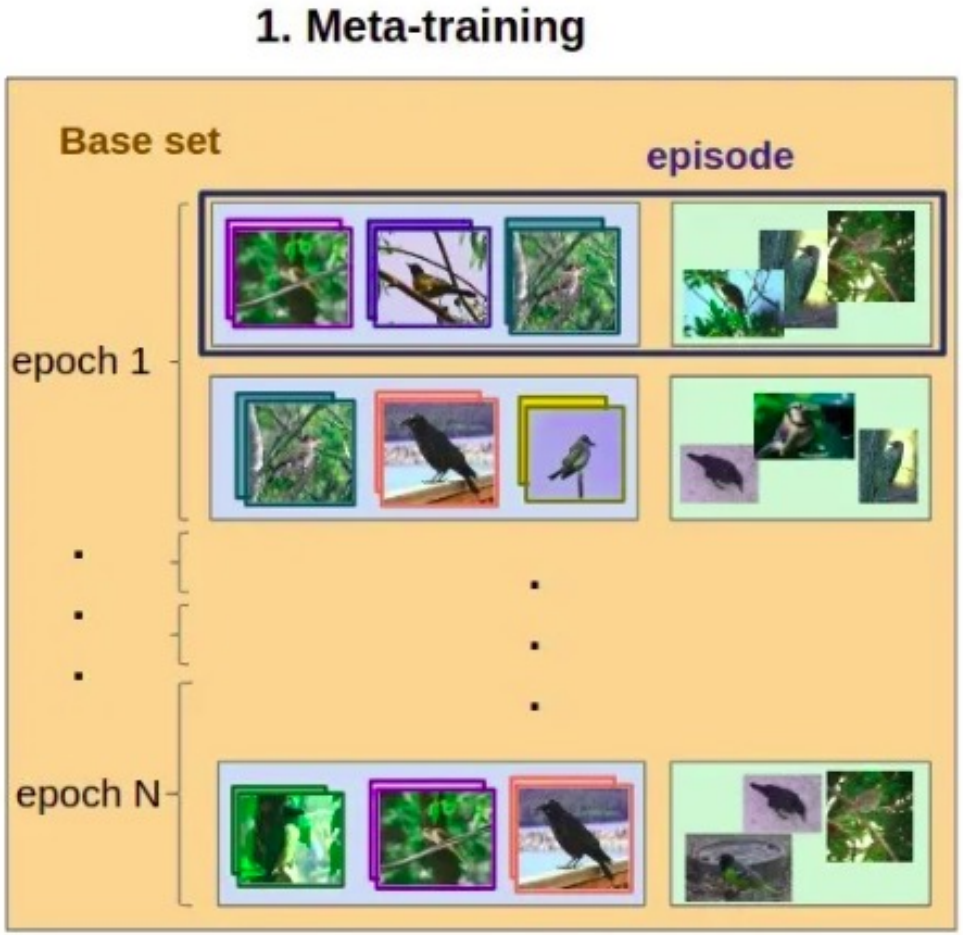
Few Shot Learning

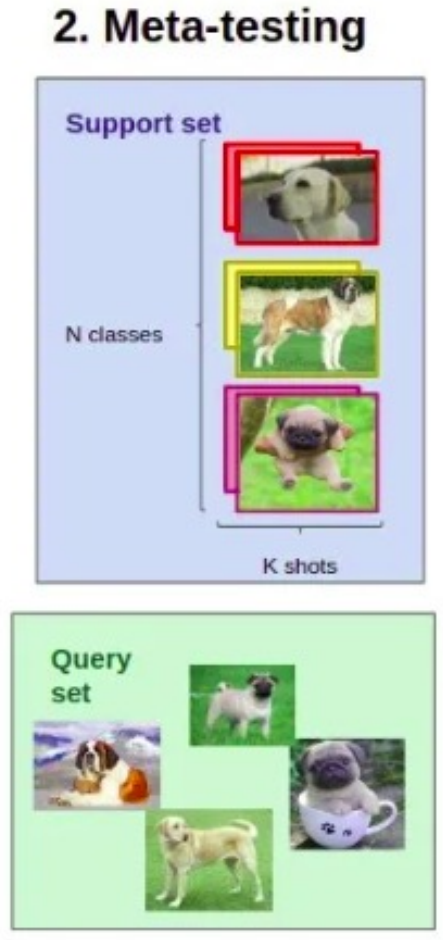
- Problem Definition
  - N Classes
  - K samples per class
  - K is usually very small
- Challenges
  - Not enough data!



Meta Learning

- Problem Definition
  - N Classes
  - K samples per class
  - New N` classes
  - Very few samples in each
  - Learn and classify!
- Challenges
  - Not enough data!
  - Catastrophic forgetting
  - Learning to learn is difficult





Approaches

Two approaches to defining a meta-learner

- Train a separate network on meta attributes:
  - Train a pattern matcher (kNN/kernel machine): Matching networks (Vinyals et al. 2016)
  - Train on Prototypes: Prototypical Networks (Snell et al. 2017)
  - Meta-Learner LSTM (Ravi & Larochelle, 2017)
- Optimization approaches:
  - ***MAML (Finn et al. 2017)***
  - Reptile (Nichol et al. 2018)
  - iMAML (Rajeswaran et al. 2019)

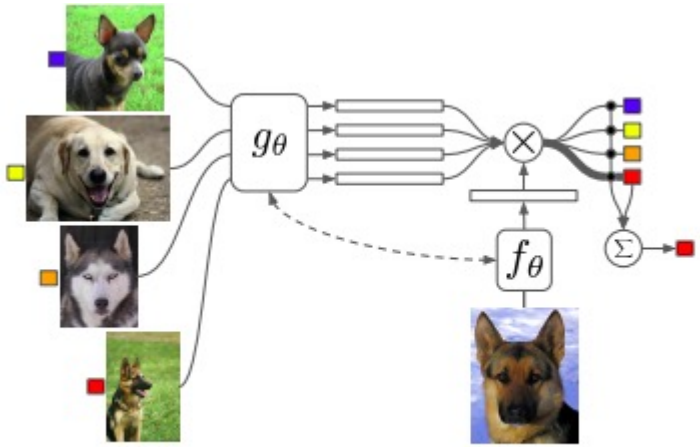
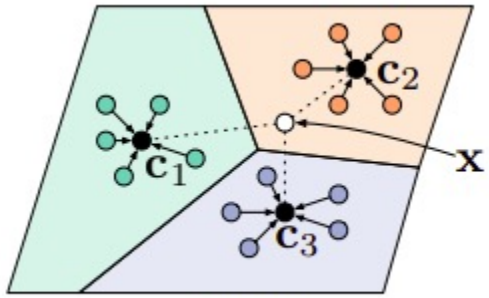
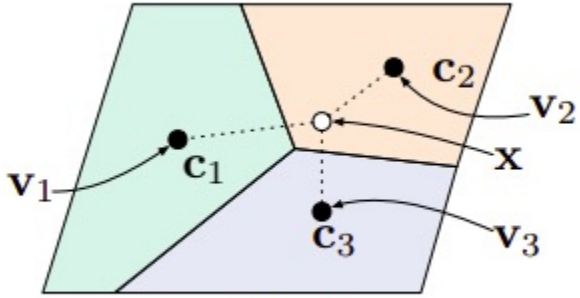


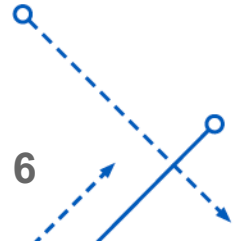
Figure 1: Matching Networks architecture



(a) Few-shot



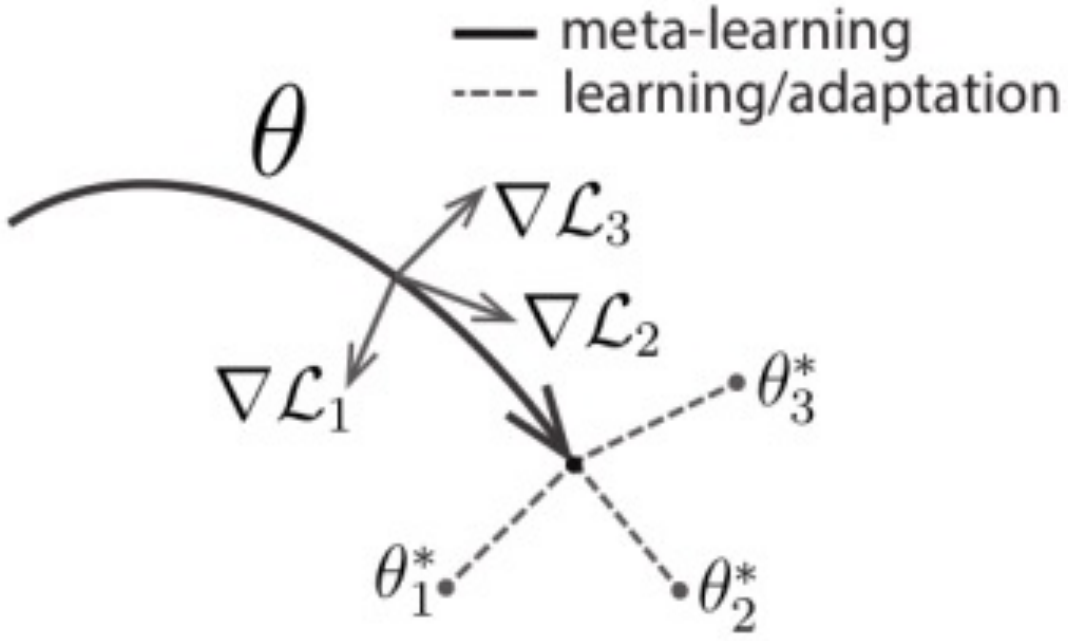
(b) Zero-shot

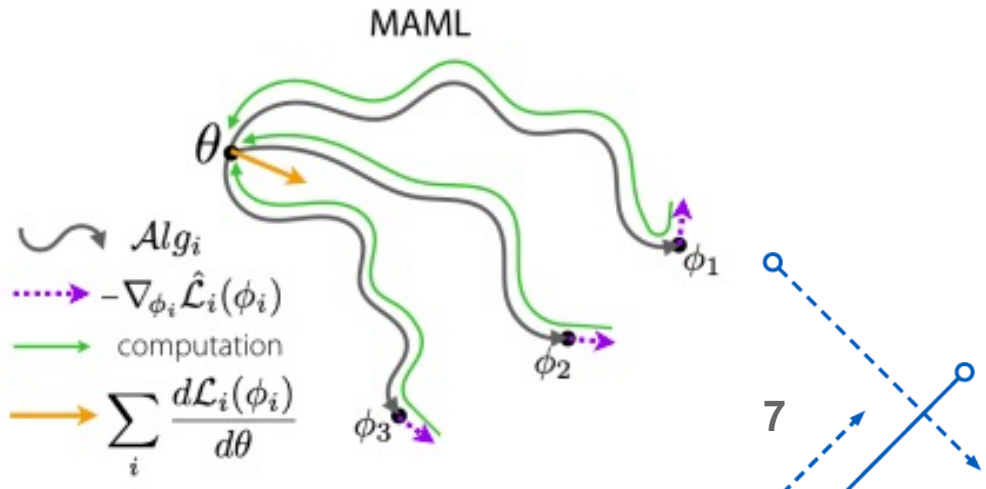


MAML

WHY IS THIS OPTIMIZATION?

- Traditional approaches to optimization learn on one task
- Learning other tasks presents a generalization problem
- Challenge is to optimize a network that translates well to other tasks
- Optimize model parameters to make them sensitive to changes in the task
- Requires a differentiable framework





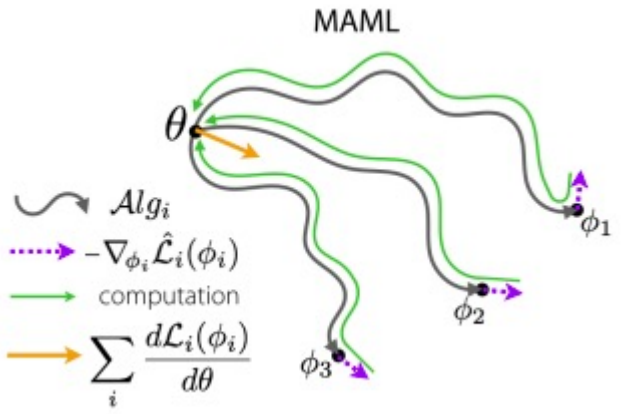
Algorithm

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**

Inner Loop: Update the model for a task from an initialization

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



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**

Gradient though gradient!!

Outer Loop: Optimize for the performance of all **inner loop** models on **all tasks**

Meta-objective:

$$\min_\theta \underbrace{\sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})}_{\text{Total loss of all updated models}} = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

Meta-update:

$$\theta \leftarrow \theta - \beta \nabla_\theta \underbrace{\sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})}_{\text{Total loss of all updated models}}$$

MAML for Few-Shot Supervised Learning

- Regression: predict the outputs of a function from only K datapoints sampled from that function, after training on many functions with similar statistical properties
- Classification: learn to classify an object only from K examples, after training on many other types of objects

How to use MAML?

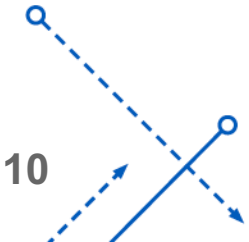
- Simply use the general framework with appropriate loss functions!

Regression: Mean-squared error (MSE)

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \| f_\phi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)} \|_2^2,$$

Classification: Cross-entropy loss

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1 - \mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)}))$$



MAML for Reinforcement Learning

- Goal: enable an agent to quickly acquire a new task policy using only a small amount of experience
- How to use MAML?
  - Use policy gradient method for a differentiable framework
  - Sample new examples with the new policy

Algorithms for FSL and RL

Algorithm 2 MAML for Few-Shot Supervised Learning

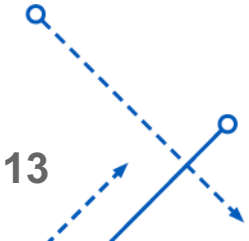
Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
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 $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
11: **end while**

Algorithm 3 MAML for Reinforcement 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: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using f_θ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample trajectories $\mathcal{D}_i' = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ 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 $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: **end while**

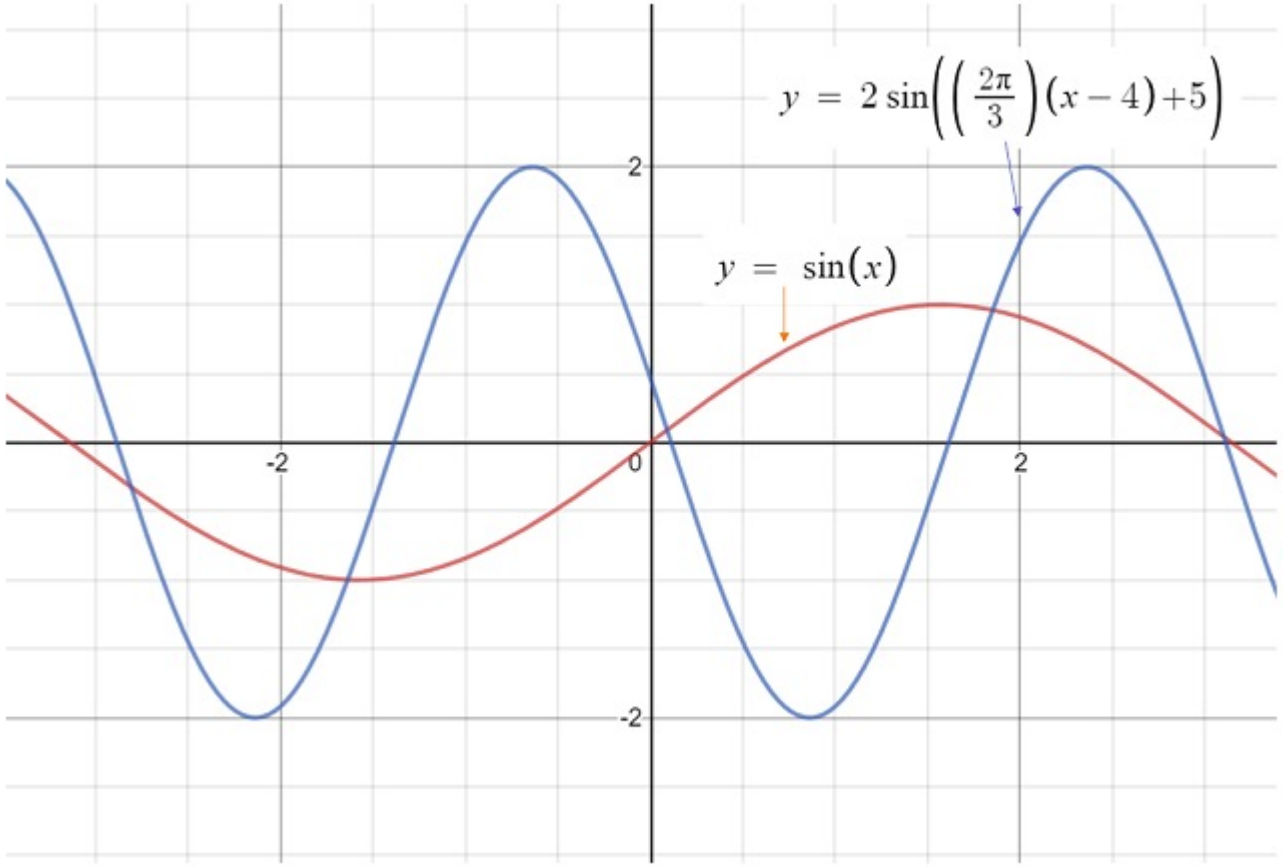
Experiments – Main Questions

- Can MAML enable fast learning of new tasks?
- Can MAML be used for meta-learning in multiple different domains?
- Can MAML models continue to improve with additional gradient updates?

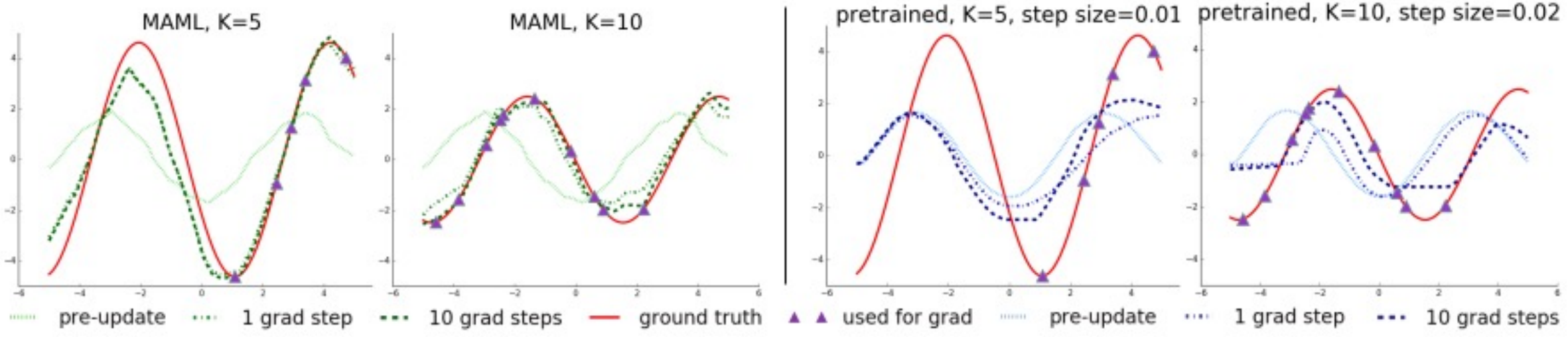


Experiments – Regression

- **Task:** Regressing to a several sine waves (varying amplitude and phase) given K data points
- Training set: Several sinusoids (tasks) not from the test set
- MAML: Meta-training on all tasks with MAML + fine-tuning on K data points
- Baseline: Pretraining on all tasks with SGD + fine-tuning on K data points
- One update for K=10 or K=5 examples, finetuned using the same samples



Experiments – Regression



- Able to extimate parts of the curve lacking data points
- Learned the periodic structure
- Right: Catastrophic forgetting and failure to extrapolate and generalize

Experiments – Classification

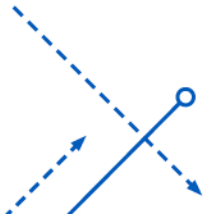
- **Task:** Few shot classification of N unseen classes with only K instances
- ***Omniglot***
  - Handwritten character classification on Omniglot
  - 20 instances of 1623 chars from 50 alphabets
- ***Mini-ImageNet***
  - Subset of ImageNet
  - 64 train, 24 val, 12 test classes

Experiments – Classification

Table 1. Few-shot classification on held-out Omniglot characters (top) and the MiniImagenet test set (bottom). MAML achieves results that are comparable to or outperform state-of-the-art convolutional and recurrent models. Siamese nets, matching nets, and the memory module approaches are all specific to classification, and are not directly applicable to regression or RL scenarios. The ± shows 95% confidence intervals over tasks. Note that the Omniglot results may not be strictly comparable since the train/test splits used in the prior work were not available. The MiniImagenet evaluation of baseline methods and matching networks is from Ravi & Larochelle (2017).

	5-way Accuracy		20-way Accuracy	
Omniglot (Lake et al., 2011)	1-shot	5-shot	1-shot	5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	**89.7 ± 1.1%**	**97.5 ± 0.6%**	–	–
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%
MAML (ours)	**98.7 ± 0.4%**	**99.9 ± 0.1%**	**95.8 ± 0.3%**	**98.9 ± 0.2%**

	5-way Accuracy	
MiniImagenet (Ravi & Larochelle, 2017)	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%
MAML, first order approx. (ours)	**48.07 ± 1.75%**	**63.15 ± 0.91%**
MAML (ours)	**48.70 ± 1.84%**	**63.11 ± 0.92%**



Experiments – RL

Task: 2D Navigation - move the point agent to a goal randomly chosen for each task

- MAML:
  - Meta-training a policy on all tasks with MAML
  - Fine-tune on sampled trajectories from pre-training
  - $$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = -\mathbb{E}_{\mathbf{x}_t, \mathbf{a}_t \sim f_\phi, q_{\mathcal{T}_i}} \left[\sum_{t=1}^{H} R_i(\mathbf{x}_t, \mathbf{a}_t) \right]$$
- Baseline 1 (pretrained): Pretraining a policy on all tasks + fine-tuning
- Baseline 2 (random): Training a policy from scratch with random initialization
- Baseline 3 (oracle): All parameters known (pos, dir, vel for agent)

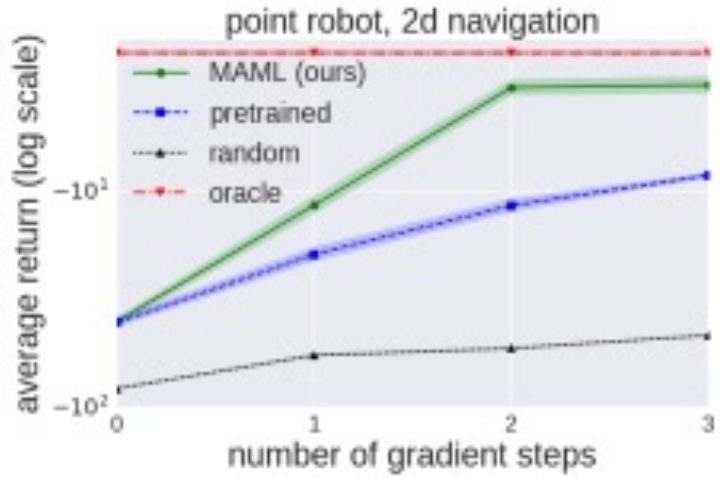
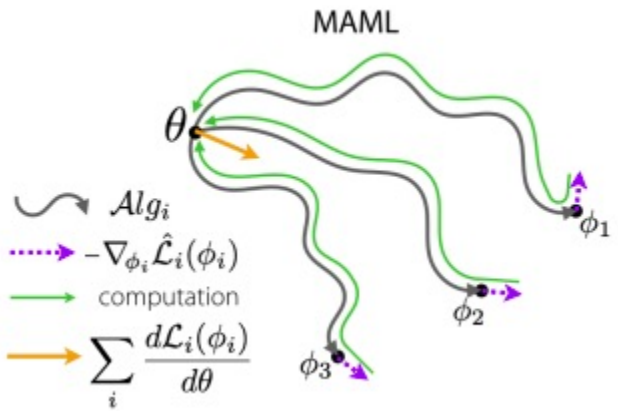


Figure 4. Top: quantitative results from 2D navigation task, Bottom: qualitative comparison between model learned with MAML and with fine-tuning from a pretrained network.

Pros and Cons

- Pros:
  - Can MAML enable fast learning of new tasks? **YES**
  - Can MAML be used for meta-learning in multiple different domains? **YES**
  - Can MAML models continue to improve with additional gradient updates? **YES**
  - Pros: Adaptable to any task that is differentiable
  - No new model for meta training
  - Generalization
- Cons:
  - Gradient through gradient makes it computationally expensive
  - Susceptible to learning rate variations
  - Not easy to train (Number of inner optimization steps to make the computation easier is an open question)



Legacy

- 7342 citations and counting!
- Used for:
  - NAS
  - Hyperparameter tuning (learning the learning rate)
  - Curriculum learning (learning easy data first)
  - You name it!
- Several of the works improve the computational efficiency

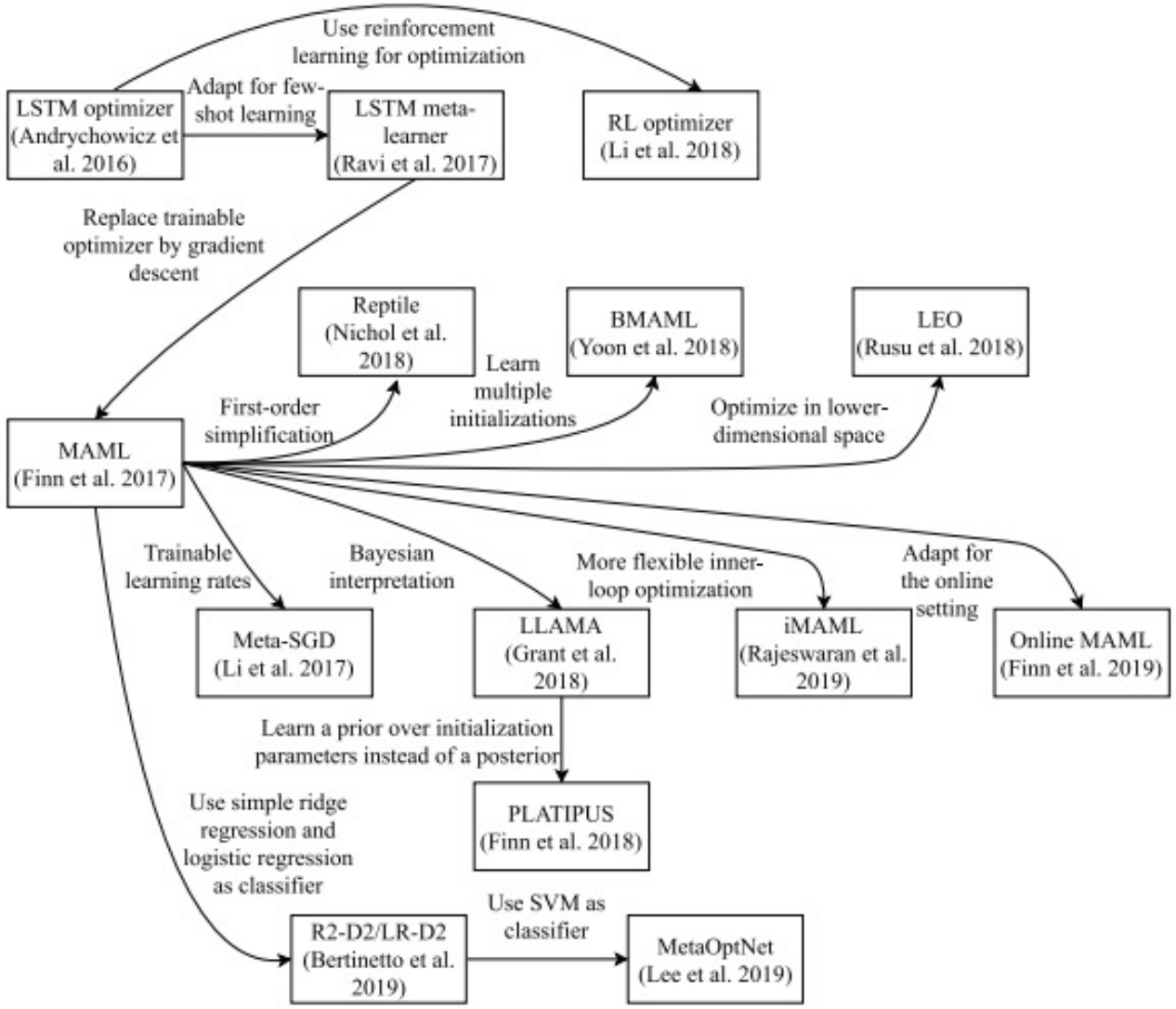


Fig. 28 The relationships between the covered optimization-based meta-learning techniques. As one can see, MAML has a central position in this network of techniques as it has inspired many other works

Thank you!