MODEL-AGNOSTIC META-LEARNING FOR FAST ADAPTATION OF DEEP NETWORKS

Presented By:

Victor Vats(50459596)

University at Buffalo The State University of New York

Outline

- Introduction
- Problem Set Up
- Algorithm
- Experiments
- Conclusion

Introduction

- A key aspect of intelligence is versatility – the capability of doing many different things.
- Current AI systems excel at mastering a single skill, such as Go, Jeopardy, or even helicopter aerobatics.
- But, when you instead ask an AI system to do a variety of seemingly simple problems, it will struggle.
- For example, a champion Jeopardy program cannot hold a conversation, and an expert helicopter controller for aerobatics cannot navigate in new, simple situations such as locating, navigating to, and hovering over a fire to put it out.
- In contrast, a human can act and adapt intelligently to a wide variety of new, unseen situations.
- The reason humans are successful at being so versatile and learning quickly is that they leverage knowledge acquired from prior experience to solve novel tasks.

Learning Algorithm

- Learn from historical data and make predictions given new examples of data.
- For example, supervised learning algorithms learn how to map examples of input patterns to examples of output patterns to address classification and regression predictive modeling problems.

Meta Learning Algorithm

- Learn from the output of learning algorithms and make a prediction given predictions made by other models.
- This means that meta-learning requires the presence of other learning algorithms that have already been trained on data.
- For example, supervised meta-learning algorithms learn how to map examples of output from other learning algorithms (such as predicted numbers or class labels) onto examples of target values for classification and regression problems.

If machine learning learns how to best use information in data to make predictions, then meta-learning or meta machine learning learns how to best use the predictions from machine learning algorithms to make predictions.

Why do we need Meta-Learning ?

- Our current AI systems can master a complex skill from scratch, but its challenging because
  - It need large datasets for training.
  - High operational costs due to many trials/experiments during the training phase.
  - Experiments/trials take a long time to find the best model which performs the best for a certain dataset.
- But if we want our agents to be able to acquire many skills and adapt to many environments, we cannot afford to train each skill in each setting from scratch.
- Instead, we need our agents to learn how to learn new tasks faster by reusing previous experience, rather than considering each new task in isolation.
- This approach of learning to learn, or meta-learning, is a key steppingstone towards versatile agents that can continually learn a wide variety of tasks throughout their lifetimes.

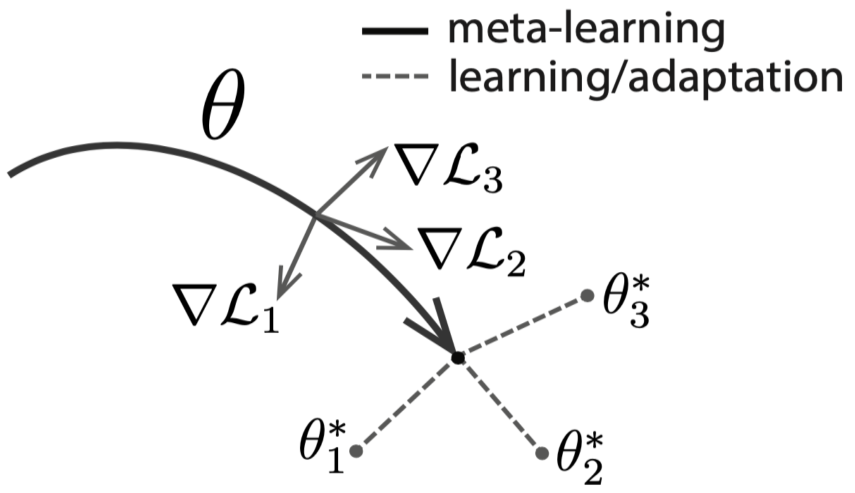
Why do we need Meta-Learning ?

- To sum up ,from a deep learning perspective, meta-learning is particularly exciting and adoptable for three reasons:
  - the ability to learn from a handful of examples.
  - learning or adapting to novel tasks quickly.
  - The capability to build more generalizable systems.

What is MAML?

MAML (Model-Agnostic Meta Learning) provides a meta-optimization objective that creates a **pre-trained model** that can easily **fine-tune** to unseen tasks!

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}\right)$$



MAML Problem Setup

Model f_θ trained on parameters θ

θ_i' denotes parameters updated for i^{th} task through gradient update steps.

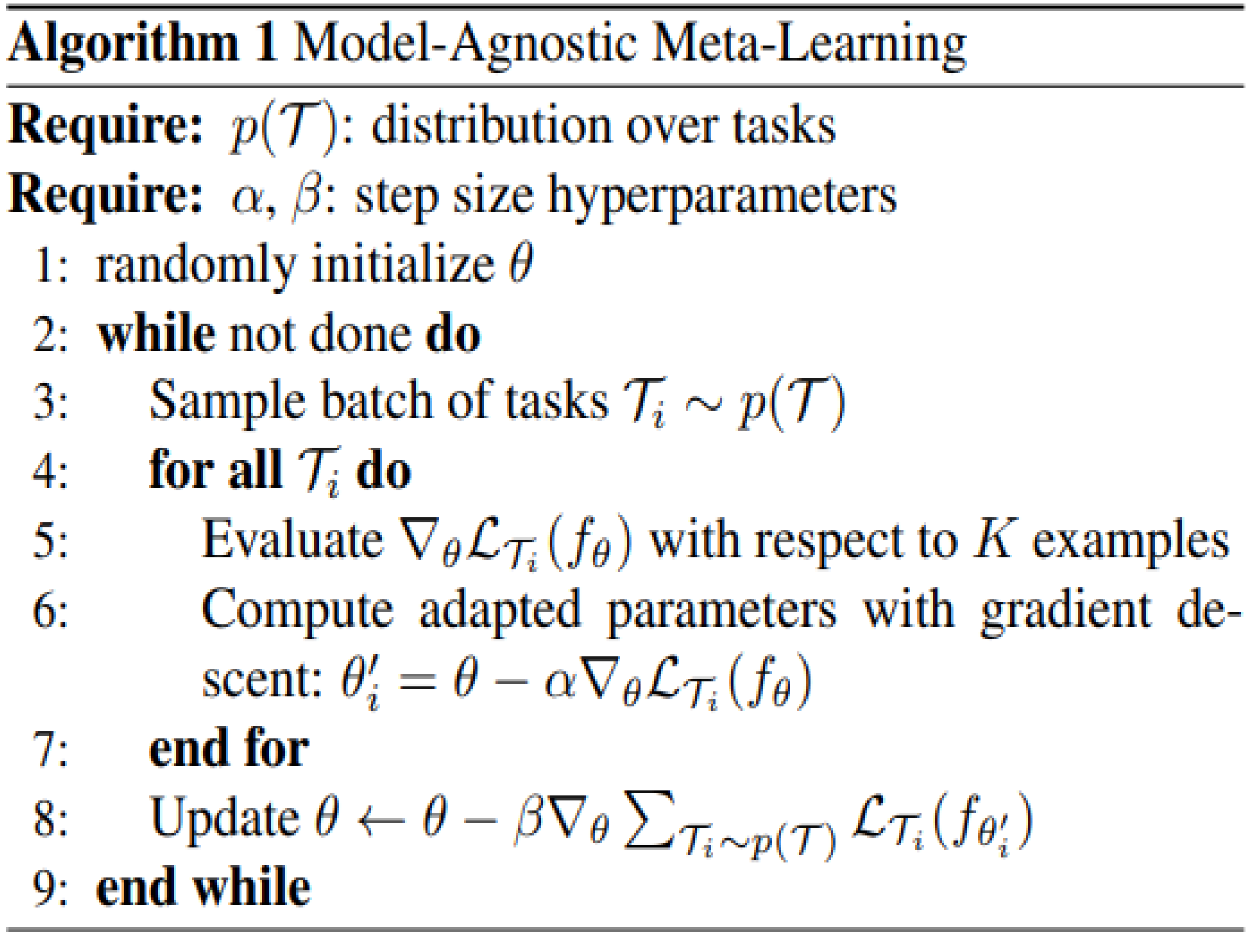
$\mathcal{T}$ is a random variable whose state space is a set of tasks.

$p(\mathcal{T})$ represents the distribution of tasks on $\mathcal{T}$.

Each task T_i has an associated loss function $\mathcal{L}_{T_i}$.

Model $f_{\theta_i'}$ is the model fine-tuned on task T_i.

MAML Algorithm



Algorithm 1 Model-Agnostic Meta-Learning

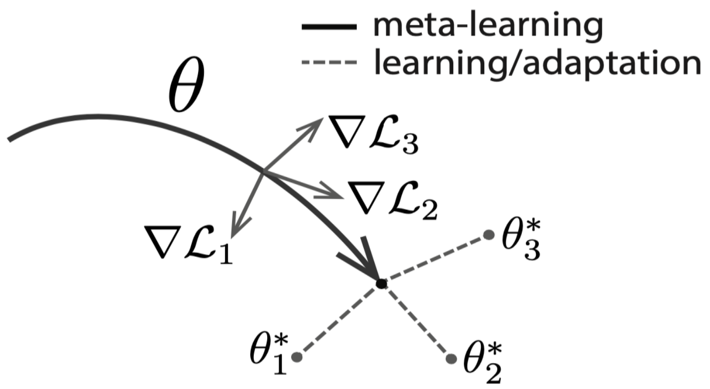
```
Require: p(T): distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize $\theta$
2: while not done do
3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4:     for all $\mathcal{T}_i$ do
5:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ 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
```

- Step 1: Randomly initialize the learner
- Step 2: Repeat the entire process from Step 2.a to Step 3 for all the episodes of the meta-training dataset (or for a certain number of epochs) until the learner converges to a good set of "meta-parameters"
  - Step 2.a: Sample a batch of episodes from the meta-training dataset
  - Step 2.b: Initialize the adapter with the learner's parameters
  - Step 2.c: While number of inner training steps is not equal to zero
    - Step 2.c.1: Train the adapter based on the support set(s) of the batch, compute the loss and the gradients, and update the adapter's parameters
  - Step 2.d: Use the updated parameters of the adapter to compute the "meta-loss" based on the query set(s) of the batch
- Step 3: Compute the "meta-gradients", followed by the "meta-parameters" based on the "meta-loss," and update the learner's parameters

MAML Gradient Update Steps

- If we're given our MAML-optimized parameters θ, we fine-tune on task Ti by doing the following gradient update rule:

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



- So, our MAML optimization meta-objective is the following:

$$\text{minimize}_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}\right)$$

- And our meta-optimization (through stochastic gradient descent) is:

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

What do we minimize with respect to?

Why can’t we create a unified set of parameters and do the following optimization problem, as opposed to the MAML one?

$$\text{minimize}_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

MAML objective

$$\boxed{\text{minimize}_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_\theta)}$$

New objective

In the MAML objective, we have a pre-trained model and then fine-tune to specific tasks. So, we minimize w.r.t. parameters that we update a few times.

The new objective is harder to fine-tune, in that the fine-tuning is not accounted for in the optimization problem. So, we get worse performance.

Benefits of MAML

- Effective for few-shot learning (only needs a few training examples per task)
- Fast (only needs 1 or a few gradient update steps)
- No extra parameters!
  - Similar algorithms that have been implemented before (e.g., RL2 by Sutskever and Abbeel) uses an entire extra RNN
- MAML optimization objective is very general and makes no assumption about structure of model (NN, CNN, RNN, etc.)
  - Only requirement is a model that optimizes on gradient descent
  - Adaptable to different ML paradigms (supervised learning, reinforcement learning, etc.)

MAML for Few-Shot Supervised Learning

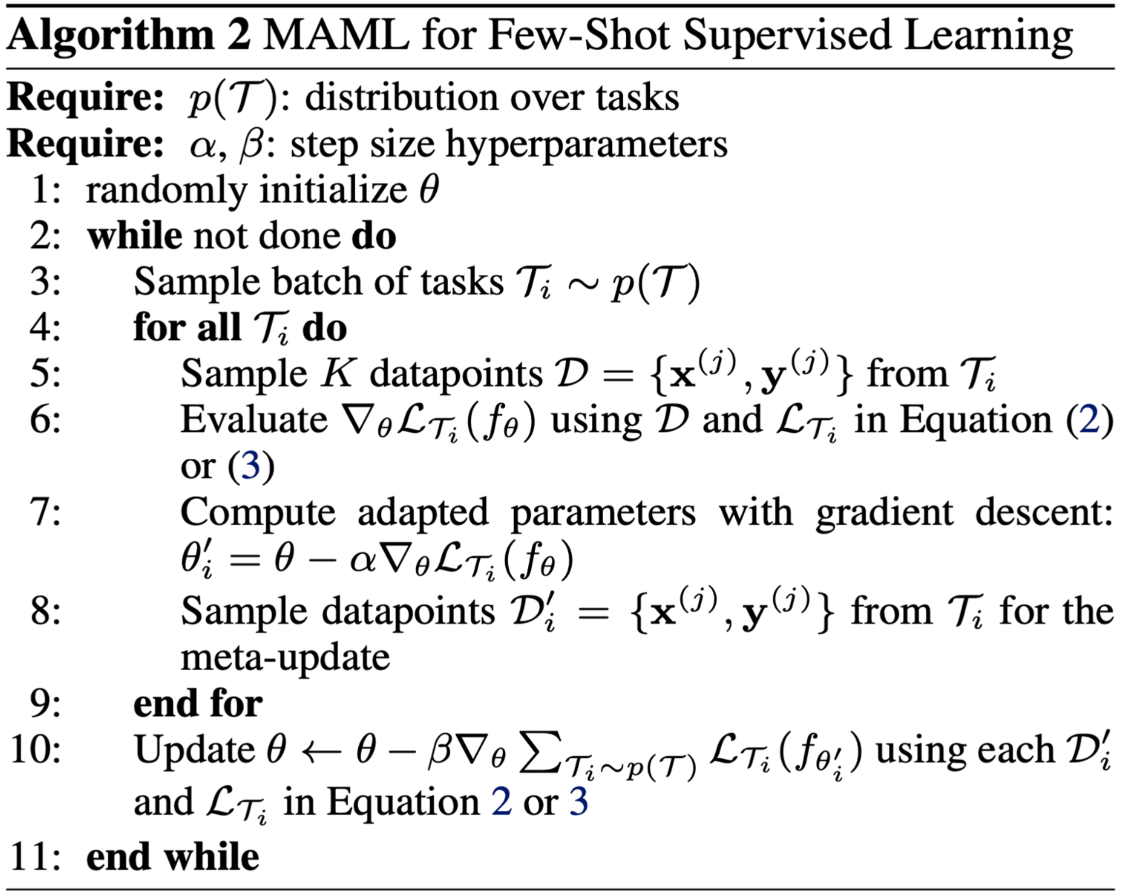
Goal: Using few data points per class, perform supervision or classification on new, unseen tasks.

Regression Objective

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

Classification Objective

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



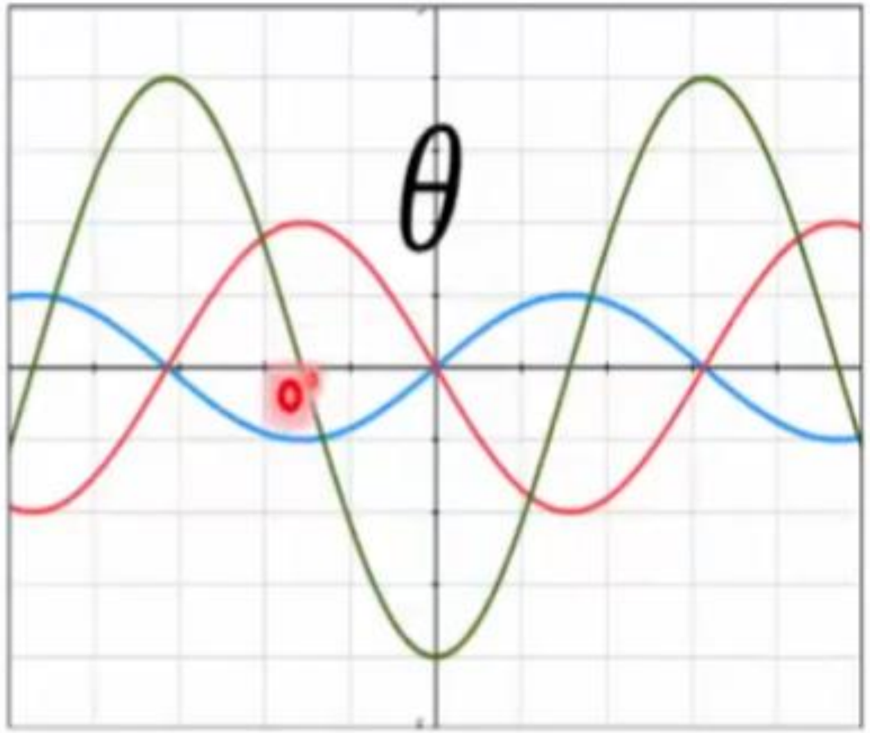
Algorithm 2 MAML for Few-Shot Supervised Learning

```
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 $\mathcal{L}_{\mathcal{T}_i}$ 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
```

University at Buffalo The State University of New York

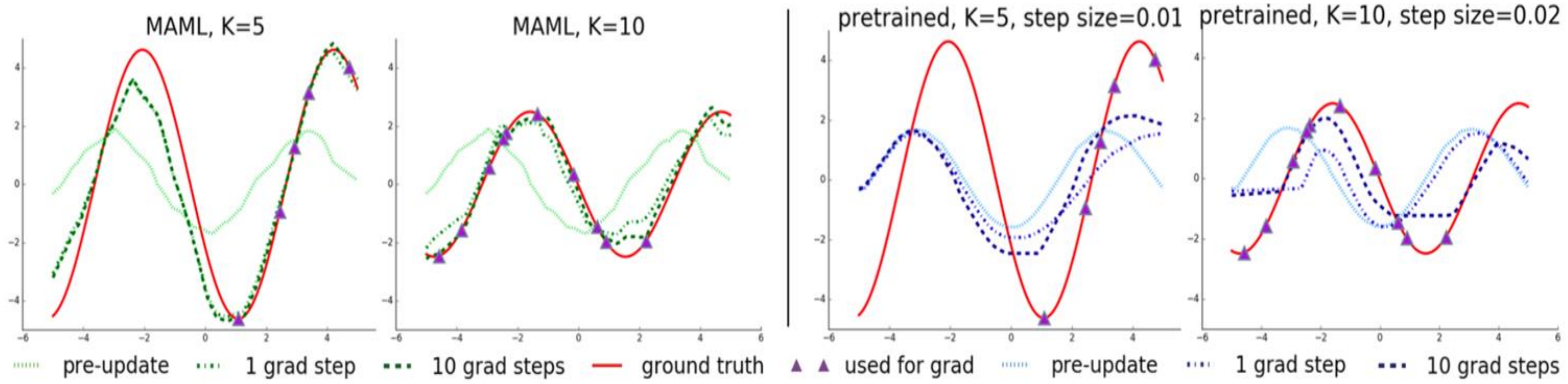
MAML Regression Experiment Results :

- Sine wave experiments
  - Meta Learning(7000000)
    - ❑Amplitude [0.1,5.0]
    - ❑Phase[0,Pi]
    - ❑K points sampled from [-5.0,5.0]
    - ❑2 fully connected layers(40 neurons) with ReLU
  - Baselines
    - ❑Pretrained Model
      - Train on all samples
      - Finetune on given sine wave during test
      - Evaluated on 600 data points
    - ❑Oracle



University at Buffalo The State University of New York

MAML Regression Experiment Results :



MAML Classification Experiment Results :

- N- way classification
  - Use N class during test with K-shot learningK shot learning
- Network Architecture
  - 4 modules
    - 3 * 3 convolutions and 64 filters
    - ReLU nonlinearity
    - 2 * 2 max pooling
- No convolution
  - 256, 128,64,64 with ReLU

- Few Shots learning benchmarks
  - Omniglot
    - 1623 characters from 50 alphabets
      - 20 instances each drawn by different person
    - Training with 1200 characters
    - Testing with 423 characters
  - MiniImageNet
    - 80 training classes
    - 20 test classes

MAML Classification Experiment Results :

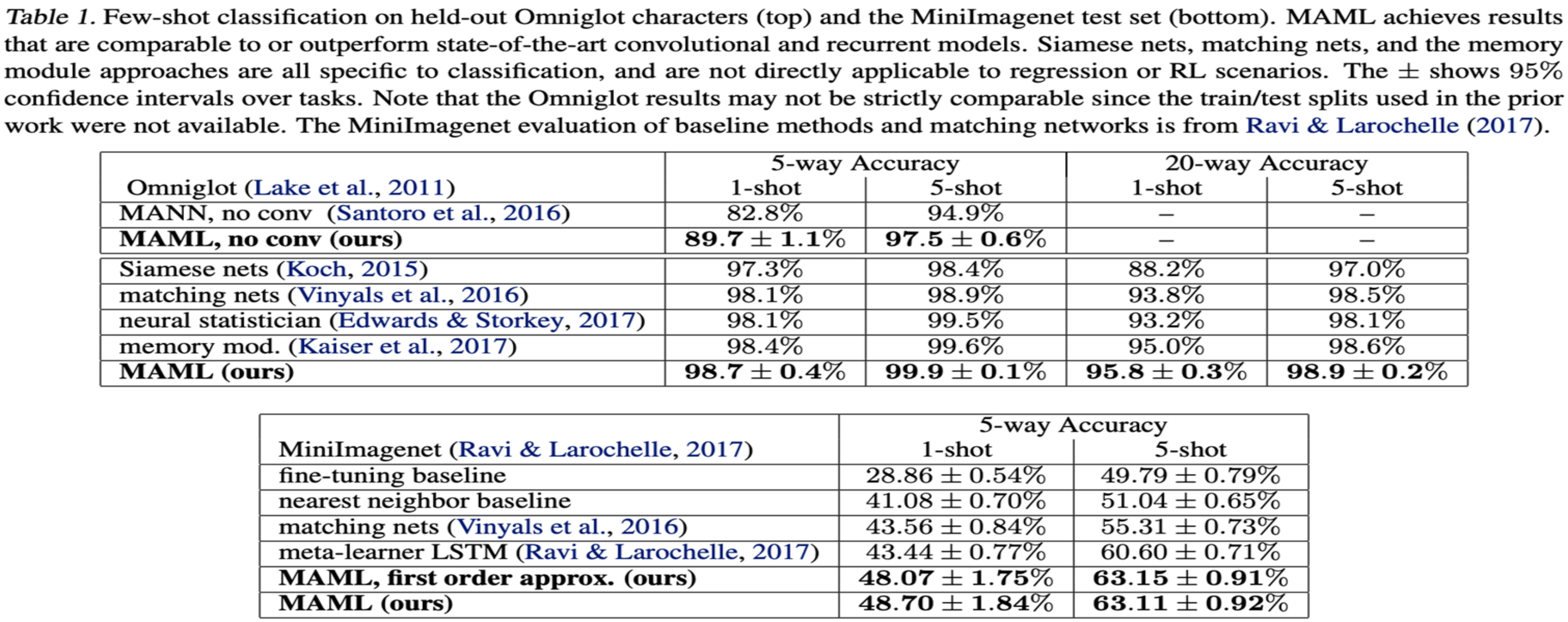


Table 1. Few-shot classification on held-out Omniglot characters (top) and the MiniImagenet test set (bottom). MAML achieves results that are comparable to or outperform state-of-the-art convolutional and recurrent models. Siamese nets, matching nets, and the memory module approaches are all specific 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%**

MAML for Reinforcement Learning

Goal: Using K trajectories/rollouts, be able to determine a policy that adapts to a new task environment.

RL Objective

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = -\mathbb{E}_{\mathbf{x}_t,\mathbf{a}_t \sim f_\phi, q_{\mathcal{T}_i}}\left[\sum_{t=1}^{H} R_i(\mathbf{x}_t, \mathbf{a}_t)\right]$$

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 $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_\theta$
            in $\mathcal{T}_i$
 6:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ 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
```

MAML on RL Experimental Results:

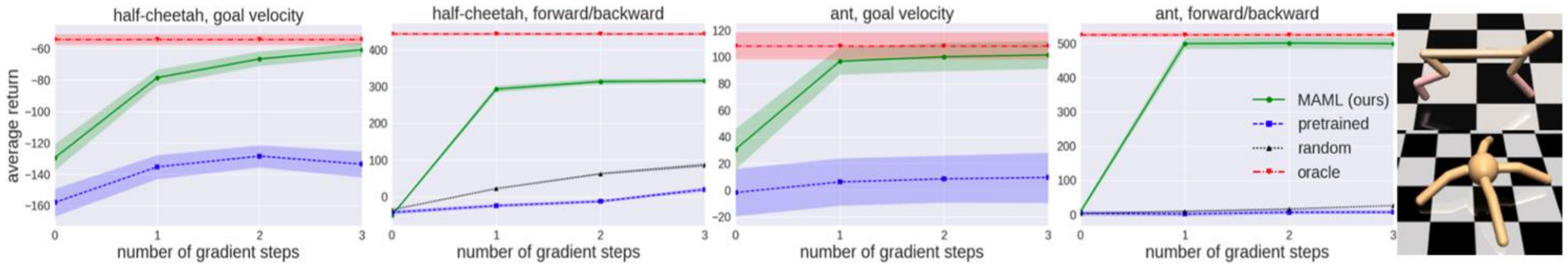


Figure 5. Reinforcement learning results for the half-cheetah and ant locomotion tasks, with the tasks shown on the far right. Each gradient step requires additional samples from the environment, unlike the supervised learning tasks. The results show that MAML can adapt to new goal velocities and directions substantially faster than conventional pretraining or random initialization, achieving good performs in just two or three gradient steps. We exclude the goal velocity, random baseline curves, since the returns are much worse (< -200 for cheetah and < -25 for ant).

Conclusion

- Multi-task learning
- Few shot learning
- MAML is an algorithm that allows us to make a pre-trained model that can quickly adapt to new tasks with a few gradient update steps
  - No new parameters
  - Operates with few training examples
- Optimization meta-objective includes the fact that we fine-tune to tasks, instead of creating an averaged-parameter model

University at Buffalo The State University of New York

References

- [Meta-Learning (fastforwardlabs.com)]
- [Model-Agnostic Meta Learning (MAML) - Google Slides]
- [An Interactive Introduction to Model-Agnostic Meta-Learning 👩‍🔬 (interactive-maml.github.io)]
- https://arxiv.org/pdf/1703.03400.pdf