# MAML for Fast Adaptation of Deep Networks

## Meta Learning

Eurekaimer

南开大学 统计与数据科学学院

2025 年 4 月 28 日

# Contents

1 背景介绍

2 方法解析

3 实验对比

4 总结

# 基本信息

会议：ICML
发表年份：2017
作者：Chelsea Finn
原文链接：点击跳转

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

**Chelsea Finn** [1] **Pieter Abbeel** [1 2] **Sergey Levine** [1]

**Abstract**

We propose an algorithm for meta-learning that is model-agnostic, in the sense that it is compatible with any model trained with gradient descent and applicable to a variety of different learning problems, including classification, regression, and reinforcement learning. The goal of meta-learning is to train a model on a variety of learning tasks, such that it can solve new learning tasks using only a small number of training samples. In our approach, the parameters of the model are explicitly trained such that a small number of gradient steps with a small amount of training data from a new task will produce good generalization performance on that task. In effect, our method trains the model to be easy to fine-tune. We demonstrate that this approach leads to state-of-the-art performance on two few-shot image classification benchmarks, produces good results on few-shot regression, and accelerates fine-tuning for policy gradient reinforcement learning with neural network policies.

## 1. Introduction

Learning quickly is a hallmark of human intelligence, whether it involves recognizing objects from a few examples or quickly learning new skills after just minutes of experience. Our artificial agents should be able to do the same, learning and adapting quickly from only a few examples, and continuing to adapt as more data becomes available. This kind of fast and flexible learning is challenging, since the agent must integrate its prior experience with a small amount of new information, while avoiding overfitting to the new data. Furthermore, the form of prior experience and new data will depend on the task. As such, for the greatest applicability, the mechanism for learning to learn (or meta-learning) should be general to the task and the form of computation required to complete the task.

In this work, we propose a meta-learning algorithm that is general and model-agnostic, in the sense that it can be directly applied to any learning problem and model that is trained with a gradient descent procedure. Our focus is on deep neural network models, but we illustrate how our approach can easily handle different architectures and different problem settings, including classification, regression, and policy gradient reinforcement learning, with minimal modification. In meta-learning, the goal of the trained model is to quickly learn a new task from a small amount of new data, and the model is trained by the meta-learner to be able to learn on a large number of different tasks. The key idea underlying our method is to train the model's initial parameters such that the model has maximal performance on a new task after the parameters have been updated through one or more gradient steps computed with a small amount of data from that new task. Unlike prior meta-learning methods that learn an update function or learning rule (Schmidhuber, 1987; Bengio et al., 1992; Andrychowicz et al., 2016; Ravi & Larochelle, 2017), our algorithm does not expand the number of learned parameters nor place constraints on the model architecture (e.g. by requiring a recurrent model (Santoro et al., 2016) or a Siamese network (Koch, 2015)), and it can be readily combined with fully connected, convolutional, or recurrent neural networks. It can also be used with a variety of loss functions, including differentiable supervised losses and non-differentiable reinforcement learning objectives.

The process of training a model's parameters such that a few gradient steps, or even a single gradient step, can produce good results on a new task can be viewed from a feature learning standpoint as building an internal representation that is broadly suitable for many tasks. If the internal representation is suitable to many tasks, simply fine-tuning the parameters slightly (e.g. by primarily modifying the top layer weights in a feedforward model) can produce good results. In effect, our procedure optimizes for models that are easy and fast to fine-tune, allowing the adaptation to happen in the right space for fast learning. From a dynamical systems standpoint, our learning process can be viewed as maximizing the sensitivity of the loss functions of new tasks with respect to the parameters: when the sensitivity is high, small local changes to the parameters can lead to

[1]University of California, Berkeley [2]OpenAI. Correspondence to: Chelsea Finn <cbfinn@eecs.berkeley.edu>.

*Proceedings of the $34^{th}$ International Conference on Machine Learning*, Sydney, Australia, PMLR 70, 2017. 

# 例子：哪些是狗

# 问题的提出

## Motivation

通常在机器学习里，我们会使用某个场景的大量数据来训练模型；然而当场景发生改变，模型就需要重新训练，这将花费很大的成本。但是对于人类而言，只需要少量样本的训练就可以很好的将所学习到的经验应用于其他任务。例如人类只需要观看过一些狗的图片，就能准确地识别狗这种生物。

这主要是因为人类能够利用不同任务间的联系，积累了对学习这类任务的经验，并用这份经验快速地对新任务进行学习，我们同样希望计算机也像人类一样，捕捉不同任务之间的相似之处，从而快速对新任务进行适应以节约成本。

这便是元学习 (Meta-Learning)，主要目标是让模型在新任务中用少量数据快速适应，从而提升现有学习算法的性能，或者学习 (诱导) 学习算法本身，因此有另一个术语“学习如何学习 (Learn-to-Learn)”。

# 元学习三大方法

- 基于模型初始化的方法（Model-Based Meta-Learning）
  - 核心思路：通过元训练优化模型的初始参数，使得该参数在新任务上微调时，损失函数下降最快。
  - 代表方法：模型无关元学习（Model-Agnostic Meta-Learning, MAML）
- 基于度量的方法（Metric-Based Meta-Learning）
  - 核心思路：构建样本的特征嵌入空间，利用度量函数（如余弦距离、欧氏距离）比较查询样本与支持样本的相似性，实现快速推理。
  - 代表方法：孪生网络（Siamese Networks）、原型网络（Prototypical Networks, ProtoNets）
- 基于优化器的方法（Optimization-Based Meta-Learning）
  - 核心思路：将优化器本身视为可学习的模型（如 RNN），通过元训练让优化器学会如何更新目标模型的参数。
  - 代表方法：元学习优化器（Meta-Learning Optimizer, MAML-Opt）

# 对比

| 方法类型 | 核心思路 | 代表方法 | MAML 的优势 |
| --- | --- | --- | --- |
| 基于初始化 | 优化初始参数 | MAML | 与模型无关，兼容任意架构 |
| 基于度量 | 构建特征嵌入空间 | 孪生网络 | 无需显式设计度量函数 |
| 基于优化器 | 学习参数更新规则 | LSTM 优化器 | 计算成本低，无需额外参数 |

# 一些基本概念

- MAML(Model-Agnostic Meta-earning)
  - 模型无关 (model-agnostic)：与模型无关的元学习，可兼容于任何一种采用梯度下降算法的模型
  - 元学习 (meta-learning)
- N-way K-shot
- Task

# N-way K-shot

N-way K-shot 是元学习（Meta-Learning）和小样本学习（Few-Shot Learning）中用于描述任务设定的核心概念，用于模拟模型在少量样本下学习新类别的能力。其本质是通过构建一个模拟的“新任务”，测试模型能否仅通过少量样本快速识别新类别。

- N-way：表示任务中包含 N 个新的类别（通常是训练阶段未见过的“新类”）。例如，5-way 任务意味着需要区分 5 个新类别（如猫、狗、鸟、鱼、马）。
- K-shot：表示每个新类别仅提供 K 个带标签的样本（称为“支持样本”）用于学习。例如，1-shot 表示每个类别只有 1 个样本，5-shot 表示每个类别有 5 个样本。

5-way 1-shot ——识别 5 个新物种，每物种仅 1 张图片

# Task

基本学习单元：区别于传统深度学习以一批一批的数据为基本单元进行学习，元学习是以一个一个的任务进行学习的。元学习系统通过在多个任务上进行训练，学习如何高效地解决这些任务，即通过在一个一个的任务上面，对任务优化，从而学习到完成这些任务所需要的知识。

二者的目的都是为了学习到需要的函数，不同的是元学习学习到的函数是用于学习函数的函数

$$\mathbf{F} \rightarrow f \rightarrow task$$

每个 Task 包含 Support set $S_i$，Query set $Q_i$

注意：选取训练集时可以在训练样本中进行反复组合，允许样本的重复，也就是某个任务的验证集可能是其他任务的训练集，但是并无影响只需要有一定差异即可.

# Contents

1 背景介绍

2 方法解析

3 实验对比

4 总结

# Key Idea

研究目标：提出一种通用且与模型无关的元学习算法，适用于各种学习问题和模型架构，通过梯度下降进行训练。

算法核心思想：训练模型的初始参数，使得在新任务上仅通过少量梯度更新就能取得良好的泛化性能，即让模型易于微调。

# Intuition

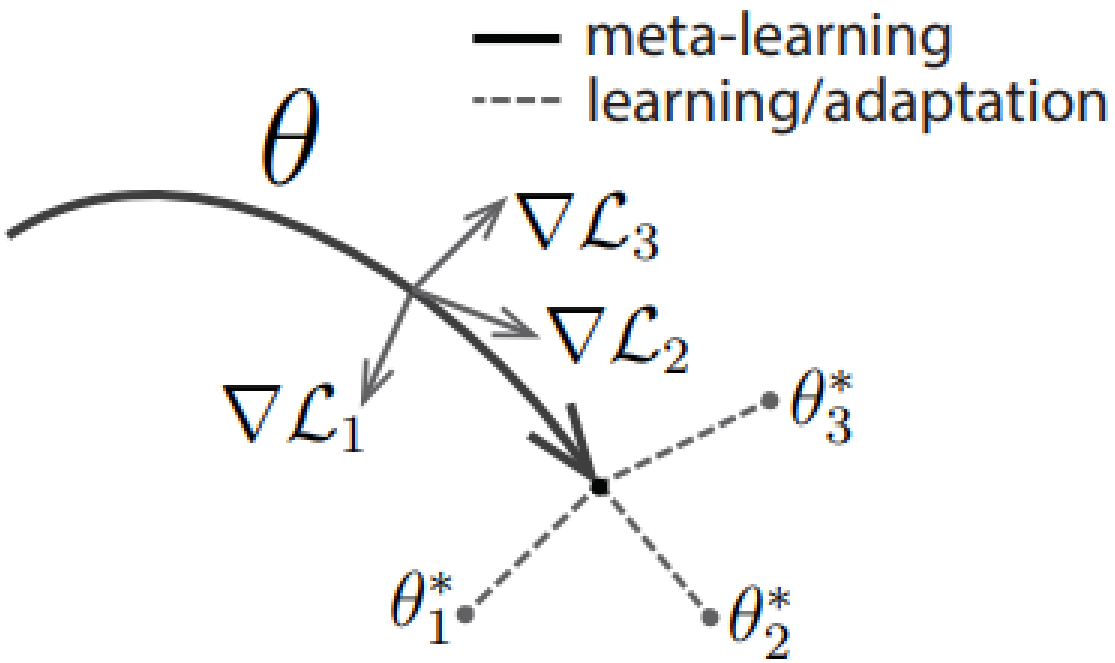



*Figure 1.* Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation $\theta$ that can quickly adapt to new tasks.

# 如何实现？

# Gradient by gradient

该算法涉及两次梯度下降：

- 内层循环（Inner Loop）
  在单个任务上快速适应
- 外层循环（Outer Loop）
  在元训练数据中更新初始参数

# 算法伪代码

**Algorithm 1** Model-Agnostic Meta-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: 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**

# 算法伪代码

**Algorithm 1** Model-Agnostic Meta-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: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples

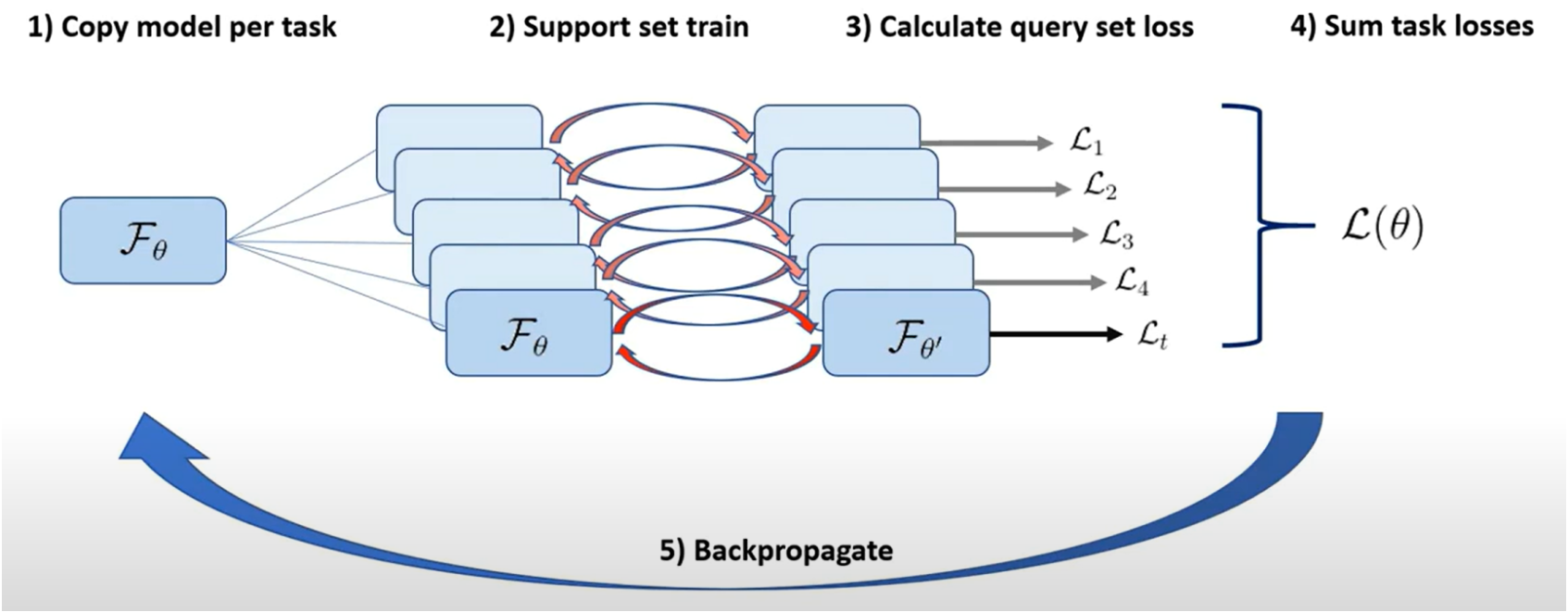
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$

7: **end for**

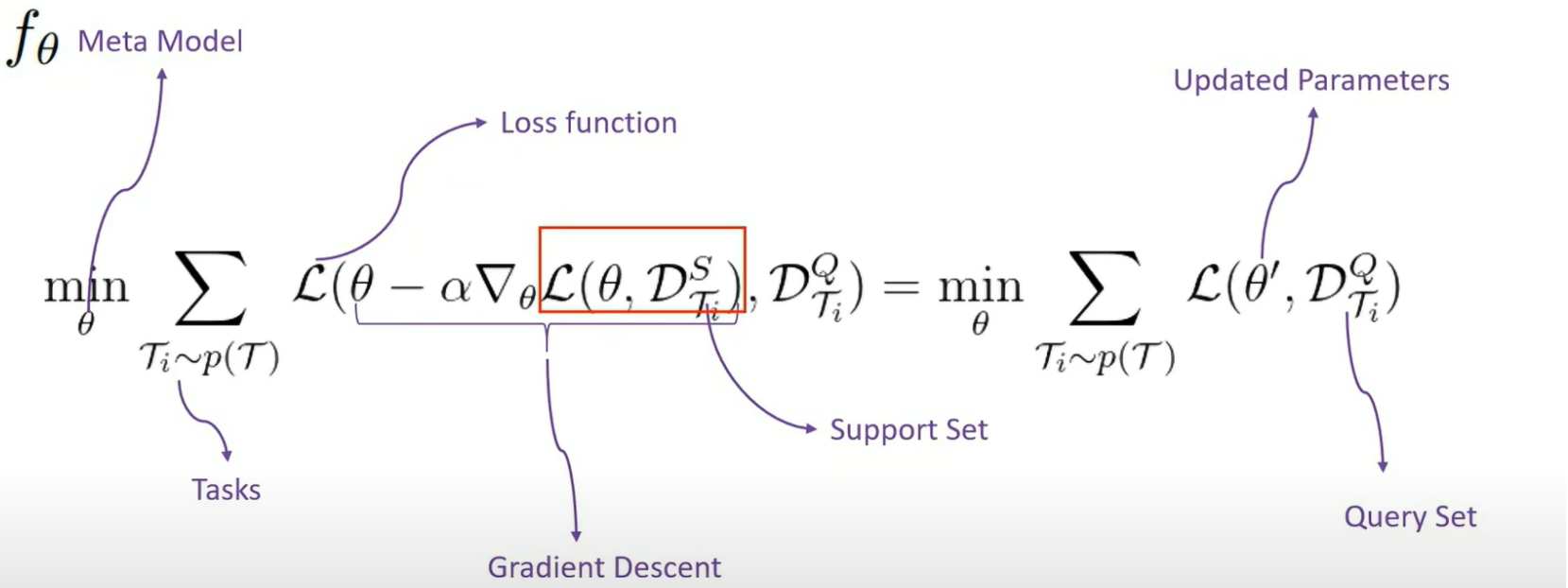
8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$

9: **end while**

# 图示


1) Copy model per task
2) Support set train
3) Calculate query set loss
4) Sum task losses
$\mathcal{F}_\theta$
$\mathcal{F}_\theta$
$\mathcal{F}_{\theta'}$
$\mathcal{L}_1$
$\mathcal{L}_2$
$\mathcal{L}_3$
$\mathcal{L}_4$
$\mathcal{L}_t$
$\mathcal{L}(\theta)$
5) Backpropagate

# 损失函数表达式


$f_\theta$ Meta Model
Loss function
Updated Parameters
$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^S_{\mathcal{T}_i}), \mathcal{D}^Q_{\mathcal{T}_i}) = \min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\theta', \mathcal{D}^Q_{\mathcal{T}_i})$
Tasks
Support Set
Gradient Descent
Query Set

# 分类与回归

## 损失函数对比

回归：MSE

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

分类: 交叉熵

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

# 算法伪代码

**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**

**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**

与初始算法并无明显不同
唯一不同的是损失函数选用 MSE 或是交叉熵

# 详细过程

两个 Require:

- 样本池 (task 分布)
- 两个超参数 (学习率)

Step1: 随机初始化参数 $\theta$

Step2: 随机采样形成 batch

Step3: 第一次梯度更新

对于每个任务 $T_i$，都进行内循环梯度更新，相当于是 copy 了一份进行更新，但是不修改原本的参数

$$\theta_i^{'} = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)$$

The step size $\alpha$ may be fixed as a hyperparameter or meta-learned. For simplicity of notation, we will consider one gradient update for the rest of this section, but using multiple gradient updates is a straightforward extension.

# 详细过程

Step4: 第二次梯度更新 (此时获得需要的参数)

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

注：这里的损失函数是在所有的 task 中的总和

braries such as TensorFlow (Abadi et al., 2016). In our experiments, we also include a comparison to dropping this backward pass and using a first-order approximation,

# 一阶近似

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

$$\nabla_\theta L(\phi) = \nabla_\theta \sum_{n=1}^{N} l^n(\hat{\theta}^n) = \sum_{n=1}^{N} \nabla_\theta l^n(\hat{\theta}^n)$$

$$\nabla_\theta l(\hat{\theta}) = \begin{bmatrix} \frac{\partial l(\hat{\theta})}{\partial \theta_1} \\ \frac{\partial l(\hat{\theta})}{\partial \theta_2} \\ \cdots \\ \frac{\partial l(\hat{\theta})}{\partial \theta_i} \\ \cdots \end{bmatrix}$$

# 一阶近似

$$\frac{\partial l(\hat{\theta})}{\partial \theta_i} = \sum_j \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \frac{\partial \hat{\theta}_j}{\partial \theta_i}$$

$$\hat{\theta} = \theta - \alpha \nabla_\theta l(\theta)$$

$$i \neq j : \frac{\partial \hat{\theta}_j}{\partial \theta_i} = -\alpha \frac{\partial l(\theta)}{\partial \theta_i \partial \theta_j} \approx 0$$

$$i = j : \frac{\partial \hat{\theta}_j}{\partial \theta_i} = 1 - \alpha \frac{\partial l(\theta)}{\partial \theta_i \partial \theta_j} \approx 1$$

# FOMAML

$$\nabla_\theta l(\hat{\theta}) = \nabla_{\hat{\theta}} l(\hat{\theta})$$

$$\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f(\theta_i')) = \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_i}(f(\theta_i')) \cdot \nabla_\theta \theta_i'$$

$$\nabla_\theta \theta_i' = I - \alpha \nabla_\theta^2 \mathcal{L}_{\mathcal{T}_i}(f_\theta) \approx I$$

极大的节省了时间，也没有丢失很多精度

# Reptile

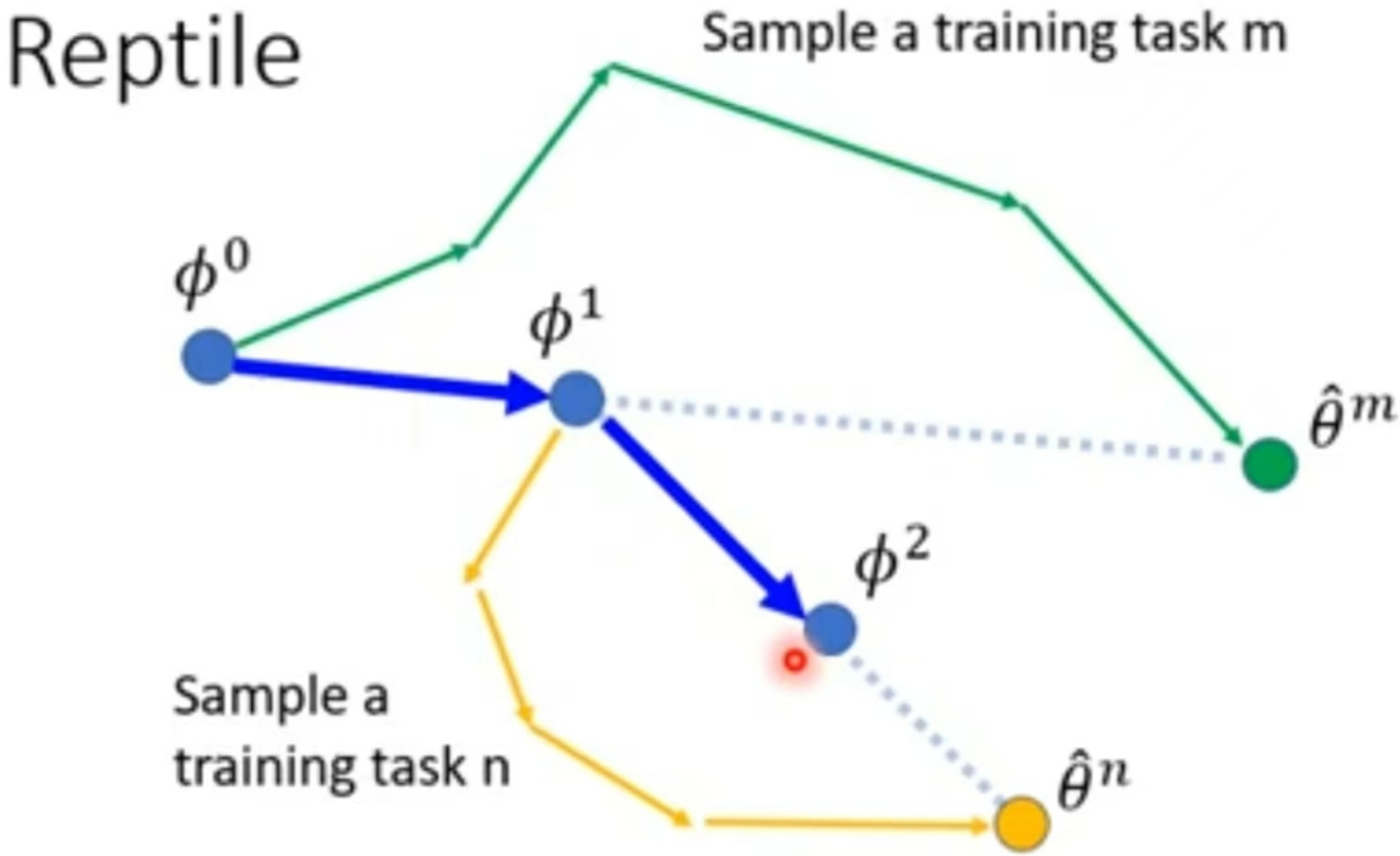
Step1：采样任务, 从任务分布 $\mathcal{T}$ 中随机采样一个任务 $T_i$。
Step2：局部更新在任务 $T_i$ 上，使用当前参数 $\theta$ 计算损失 $\mathcal{L}_{T_i}(f_\theta)$，并进行 $k$ 次梯度更新（$k$ 通常是一个较小的正整数），得到更新后的参数 $\theta'$。
第 $j$ 次更新的公式为：

$$\theta^{(j+1)} = \theta^{(j)} - \alpha \nabla_{\theta^{(j)}} \mathcal{L}_{T_i}(f_{\theta^{(j)}}) \tag{1}$$

其中 $\theta^{(0)} = \theta$，经过 $k$ 次更新后得到 $\theta' = \theta^{(k)}$。
Step3：全局更新使用更新后的参数 $\theta'$ 与原始参数 $\theta$ 的差值来更新全局参数 $\theta$：

$$\theta \leftarrow \theta + \beta(\theta' - \theta) \tag{2}$$

其中 $\beta$ 是学习率。


Reptile
Sample a training task m
$\phi^0$
$\phi^1$
$\hat{\theta}^m$
$\phi^2$
Sample a
training task n
$\hat{\theta}^n$

# Reptile

可以将其理解为最小化以下目标函数：

$$\min_{\theta} \mathbb{E}_{T_i \in \mathcal{T}} d(\theta', \theta) \tag{3}$$

其中 $d(\cdot, \cdot)$ 是某种距离度量，通常可以使用欧几里得距离，在这种情况下，目标函数等价于最小化

$$\min_{\theta} \mathbb{E}_{T_i \in \mathcal{T}} \|\theta' - \theta\|^2 \tag{4}$$

通过不断重复上述步骤，模型可以学习到一个通用的初始参数 $\theta$，使得在新任务上能够快速适应。

# Contents

1 背景介绍

2 方法解析

3 实验对比

4 总结

# 对比实验

- sinusoid regression,
- Omniglot classification
- MiniImagenet classification

# Regression

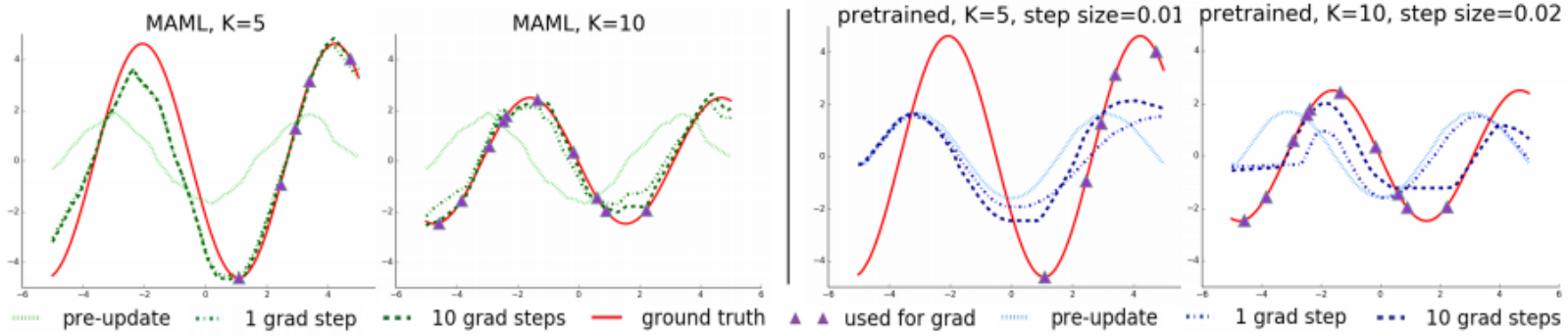



*Figure 2.* Few-shot adaptation for the simple regression task. Left: Note that MAML is able to estimate parts of the curve where there are no datapoints, indicating that the model has learned about the periodic structure of sine waves. Right: Fine-tuning of a model pretrained on the same distribution of tasks without MAML, with a tuned step size. Due to the often contradictory outputs on the pre-training tasks, this model is unable to recover a suitable representation and fails to extrapolate from the small number of test-time samples.

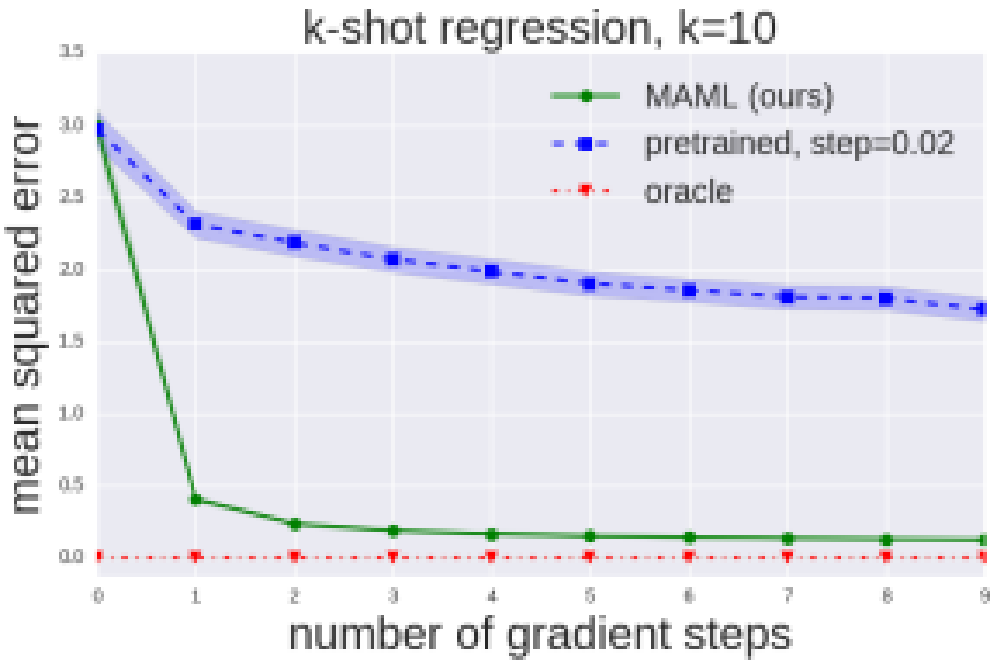



*Figure 3.* Quantitative sinusoid regression results showing the learning curve at meta test-time. Note that MAML continues to improve with additional gradient steps without overfitting to the extremely small dataset during meta-testing, achieving a loss that is substantially lower than the baseline fine-tuning approach.

# 对比图

| Omniglot (Lake et al., 2011) | 5-way Accuracy 1-shot | 5-way Accuracy 5-shot | 20-way Accuracy 1-shot | 20-way Accuracy 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%** |

| MiniImagenet (Ravi & Larochelle, 2017) | 5-way Accuracy 1-shot | 5-way Accuracy 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 为什么会好

- Rapid learning 训练出的初始参数足够好，有很好的学习能力
- √ Feature reuse 训练的参数离最终好的结果足够近，可以快速适应

Reference: Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML

链接：https://arxiv.org/pdf/1909.09157

# MAML 有哪些问题

主要存在的问题：

- 训练不稳定
- 计算成本高 (计算二阶导数，inner loop 耗时)
- 超参数敏感，依赖手动调参
- 泛化性能受限

一些改进的思路 (MAML++) 可以参考:

Reference：How to train your MAML

链接：https://arxiv.org/pdf/1810.09502

# Contents

1 背景介绍

2 方法解析

3 实验对比

4 总结

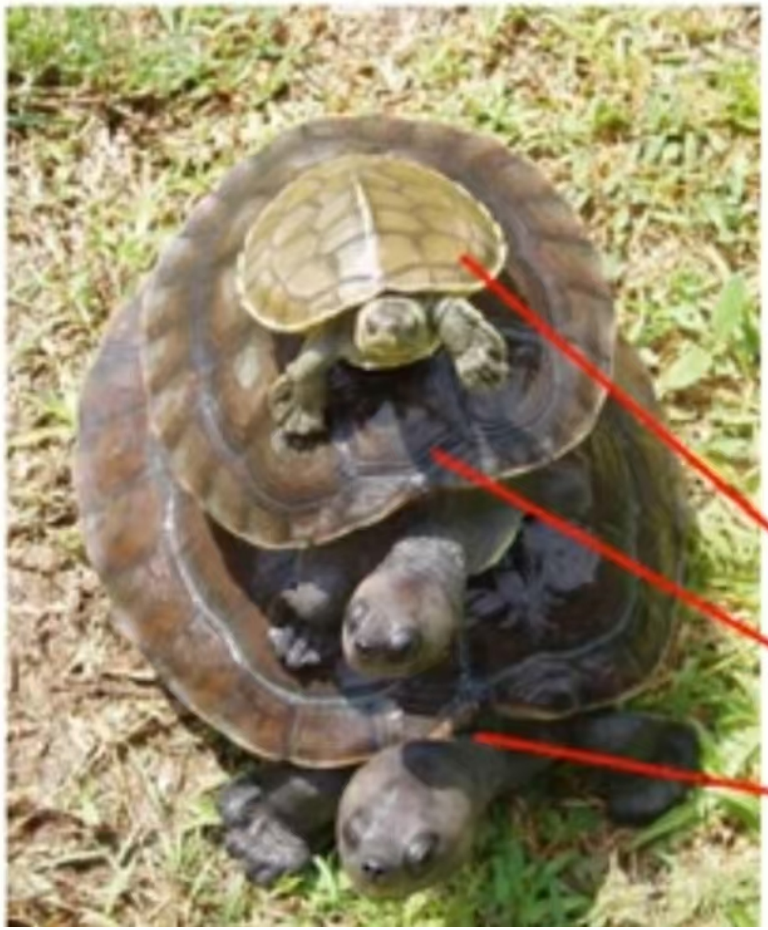
# Pros and cons

优点

- 简单优雅，容易实现。
- 为元参数估计提供了一种完全可微的方法（无需复杂的强化学习方法或遗传算法）。
- 具有“无关性”，可轻松适配多种场景（如分类、回归、强化学习），泛用性广。

缺点

- 不稳定（涉及内 / 外环循环，需对多个任务求平均）。
- 高阶导数计算成本高，导致训练时间更长。



- We learn the initialization parameter $\phi$ by gradient descent
- What is the initialization parameter $\phi^0$ for initialization parameter $\phi$?

Learn

Learn to learn

Learn to learn to learn

# 遇到的问题

- 缺少复现实验
- 电脑性能不足 (没有 N 卡, 没有 CUDA)

# Reference


- MAML for Fast Adaptation of Deep Networks
- Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML
- How to train your MAML
- 机器学习-2021-李宏毅
- CS329P-李沐
- 原作者实现 (Tensorflow)：https://github.com/cbfinn/maml
- Pytorch 版本实现：https://github.com/dragen1860/MAML-Pytorch

Thanks for listening.