Meta-Learning

EE807: Recent Advances in Deep Learning

Lecture 17

Slide made by

Yunhun Jang, Hyungwon Choi, and Hankook Lee

KAIST EE

Table of Contents

What is Meta-Learning?

- Definition from Wikipedia:

Meta learning is a subfield of machine learning where automatic learning algorithms are applied on metadata ..about machine learning experiments. As of 2017 the term had not found a standard interpretation, however the main goal is to use such metadata to understand how automatic learning can become flexible in solving learning problems, hence to improve the performance of existing learning algorithms or to learn (induce) the learning algorithm itself, hence the alternative term **learning to learn**..

- Meta learning = "**Learning to learn**"
- All kinds of learning algorithms that learns to improve the learning process itself

- Let's see an example

- An example from CUB-200 dataset: *American goldfinch*

American goldfinch

From Wikipedia, the free encyclopedia

The **American goldfinch** (*Spinus tristis*) is a small North American bird in the finch family. It is migratory, ranging from mid-Alberta to North Carolina during the breeding season, and from just south of the Canada-United States border to Mexico during the winter.

The only finch in its subfamily to undergo a complete molt, the American goldfinch displays sexual dimorphism in its coloration; the male is a vibrant yellow in the summer and an olive color during the winter, while the female is a dull yellow-brown shade which brightens only slightly during the summer. The male displays brightly colored plumage during the breeding season to attract a mate.

The American goldfinch is a granivore and adapted for the consumption of seedheads, with a conical beak to remove the seeds and agile feet to grip the stems of seedheads while feeding. It is a social bird, and will gather in large flocks while feeding and migrating. It may behave territorially during nest construction, but this aggression is short-lived. Its breeding season is tied to the peak of food supply, beginning in late July, which is relatively late in the year for a finch. This species is generally monogamous, and produces one brood each year.

Human activity has generally benefited the American goldfinch. It is often found in residential areas, attracted to bird feeders which increase its survival rate in these areas. Deforestation also creates open meadow areas which are its preferred habitat.

Contents [hide]

American goldfinch

Male American goldfinch in spring plumage

* source : https://en.wikipedia.org/wiki/American_goldfinch

What is Meta-Learning?

- Which is *American goldfinch*?





* source : http://www.vision.caltech.edu/visipedia/CUB-200.html

What is Meta-Learning?

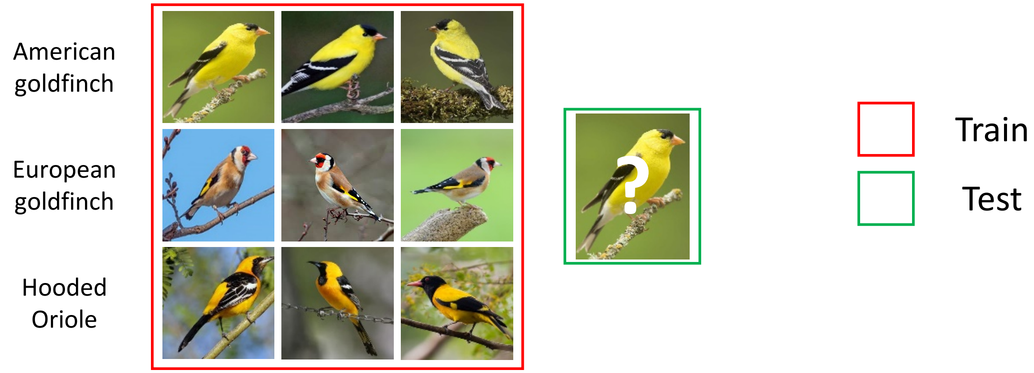
- Which is *American goldfinch*?

- Humans can quickly learn "**unseen**" classes with **small number of examples**
  - Since we have learned prior knowledge about visual representations
  - This kind of problem is called **"1-shot/few-shot" classification** problem

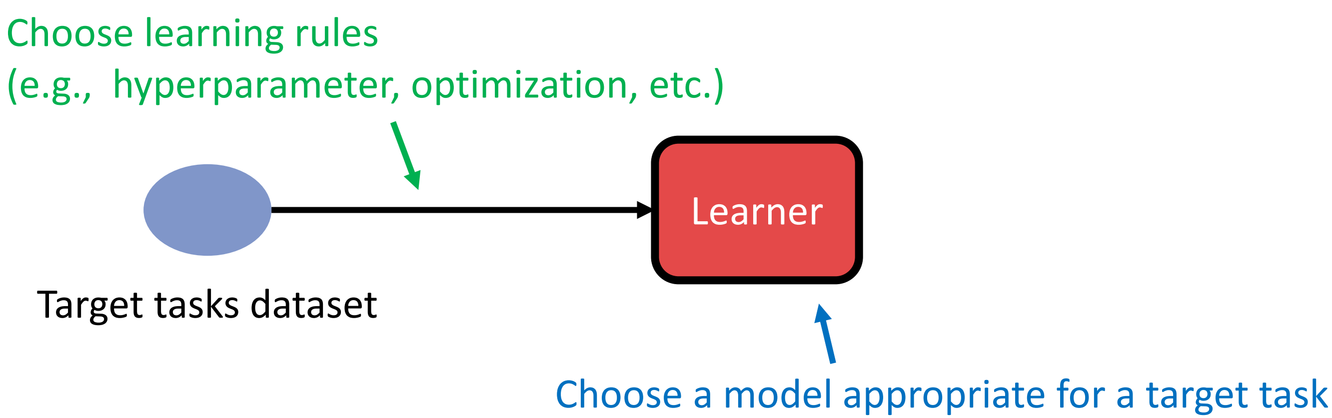
- Meta-learning: "**Learning to learn**" in order to generalize well to **unseen** tasks

* source : http://www.vision.caltech.edu/visipedia/CUB-200.html

- **Base learning** : How to learn a model to classify different classes of birds?

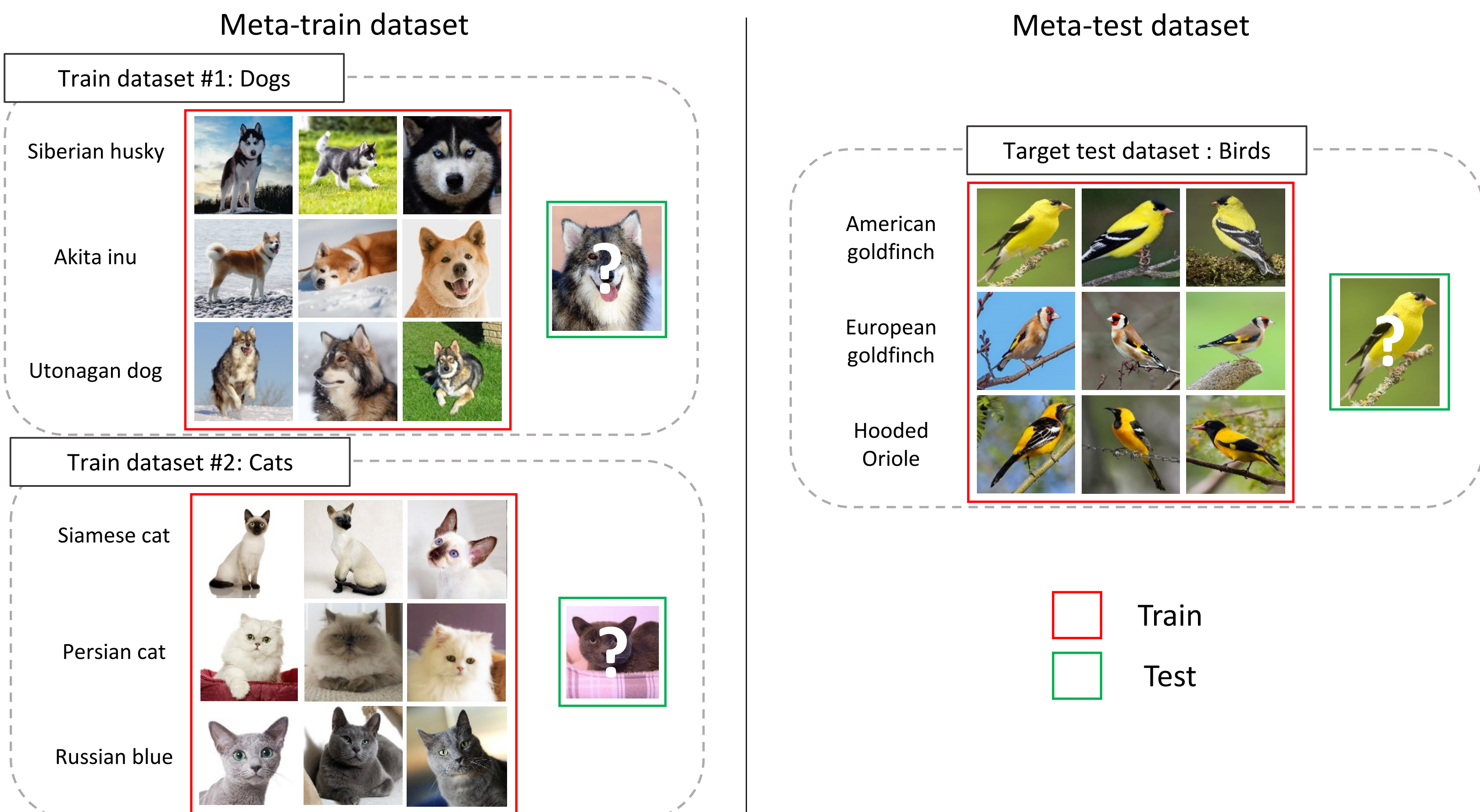


- Goal : Learn a mapping $f : \mathbf{x} \rightarrow y$ from input image $\mathbf{x}$ to output (label) y
  - Choose a **learner** (e.g., a neural network) and **learning strategies** (e.g., SGD)
  - Generally difficult when number of training samples are **very small**



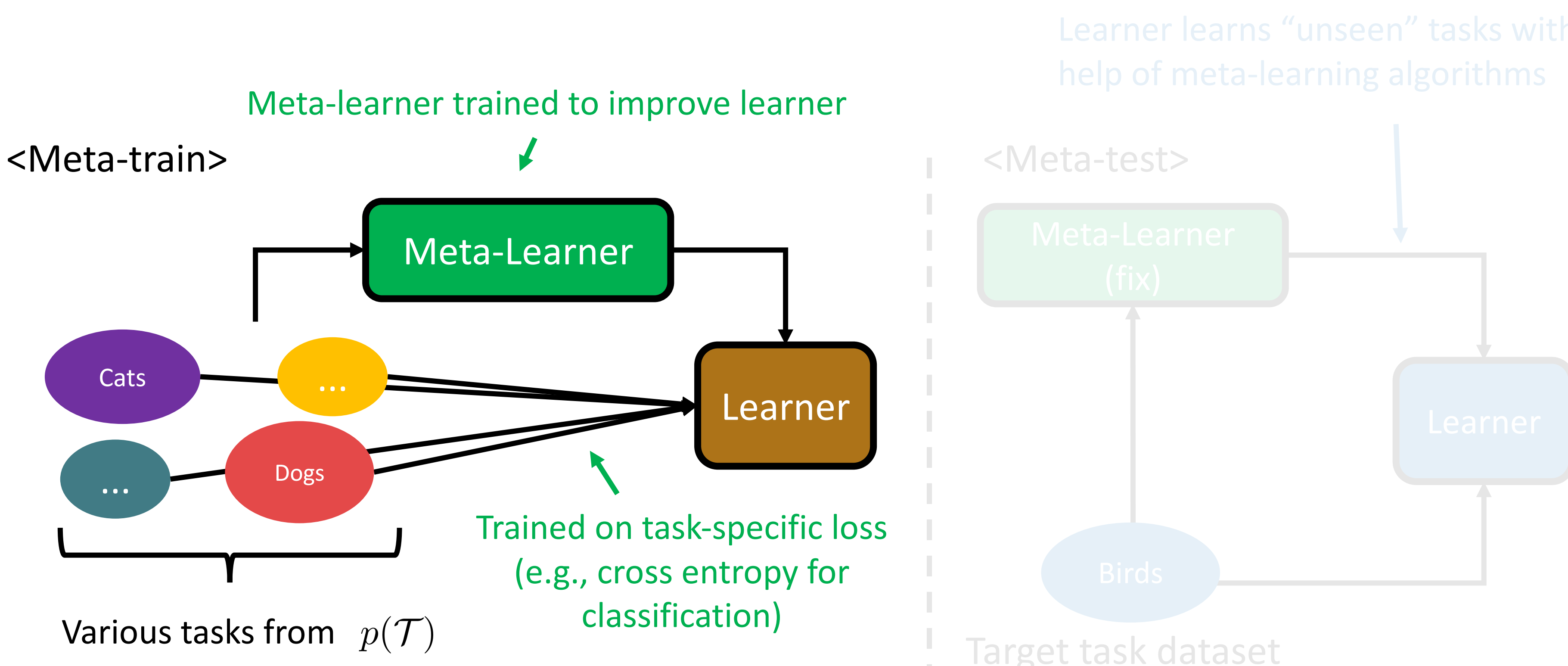
Base Learning vs. Meta-Learning

- In **meta-learning**, we focus on learning the learning rules
    - Consider each **dataset as a data sample**
    - **Learn *patterns* across tasks**
        - So that the the model can **generalize** well to possibly “unseen” tasks

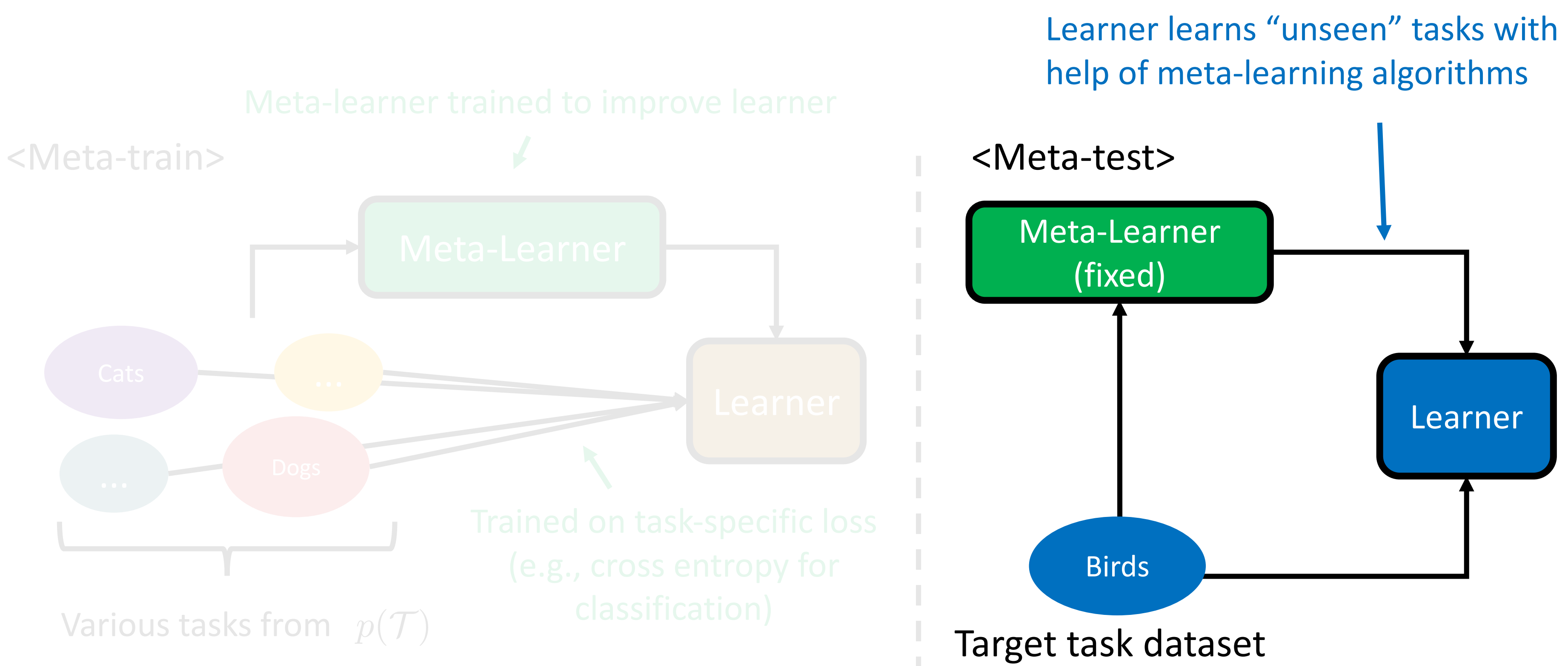


* source : https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html

- In **meta-learning**, we focus on learning the learning rules
  - Consider each **dataset as a data sample**
  - **Learn *patterns* across tasks**
    - So that the the model can **generalize** well to possibly “unseen” tasks
    - “**Learning to learn**” that works well on any task from the distributions of tasks



- In **meta-learning**, we focus on learning the learning rules
  - Consider each **dataset as a data sample**
  - **Learn** *patterns* **across tasks**
    - So that the the model can **generalize** well to possibly “unseen” tasks
    - “**Learning to learn**” that works well on any task from the distributions of tasks



- Most meta-learning algorithms consist of **two levels of learning** (or *loops*)
  - **Inner loop**: optimizes the *base learner* (e.g., classifier)
    - **Parameters** θ : parameters of the base learner
    - **Objective**: $\mathcal{L}_{\mathtt{io}}(\theta|\phi)$ (e.g., cross entropy for classification)

Algorithm 1 Common meta-learning algorithm

while not done **do**
    for $t = 1, \cdots, T$ **do**
        Optimize parameters θ of learner f_θ
        $\theta^{(t+1)} \leftarrow \theta^{(t)} - \nabla_{\theta^{(t)}}\mathcal{L}_{\mathtt{io}}$
    end for
    Optimize meta-parameters ϕ
    $\phi \leftarrow \phi - \nabla_\phi \mathcal{L}_{\mathtt{mo}}$
end while

Inner loop

Meta-Learning in More Formal Definition

- Most meta-learning algorithms consist of **two levels of learning** (or *loops*)
  - **Inner loop**: optimizes the *base learner* (e.g., classifier)
    - **Parameters** θ : parameters of the base learner
    - **Objective**: $\mathcal{L}_{\mathtt{io}}(\theta|\phi)$ (e.g., cross entropy for classification)
  - **Outer loop** (meta-training loop): optimizes *the meta-learner*
    - **Meta-parameters** ϕ : parameters to learn the learning rule (e.g., how much to update θ)
    - **Meta-objective** $\mathcal{L}_{\mathtt{mo}}(\theta, \phi)$: performance of the base learner on the new task
    - **Meta-optimization**: adjusting ϕ so that the inner loop perform well on $\mathcal{L}_{\mathtt{mo}}$

Algorithm 1 Common meta-learning algorithm

```
while not done do
    for t = 1, ..., T do
        Optimize parameters θ of learner f_θ
        θ^(t+1) ← θ^(t) − ∇_{θ^(t)} L_io
    end for
    Optimize meta-parameters φ
    φ ← φ − ∇_φ L_mo
end while
```

Inner loop

Outer loop

Applications of Meta-Learning

- Recently meta-learning is applied to many areas such as
  - Hyperparameter optimization
  - Neural network architecture search
  - Reinforcement learning
  - **Learning model initialization**
    - Overcome difficulties of few-shot learning (e.g., overfitting caused by small # of samples)
  - **Learning optimizers**
    - Instead of using hand-crafted optimizer (e.g., SGD, ADAM), learning the optimizers

In this lecture, we will focus on these two applications

Table of Contents

Learning Good Initialization for Few-Shot Learning

- **Few-shot learning** tackles limited-data scenario
  - One way to overcome the lack of data is **initialization**

- Common initialization method: pre-train with ImageNet and fine-tune
  - (+) Generally works very well on various tasks
  - (-) **Not work** when one has **only** a small number of examples (1-shot, 5-shot, etc.)
  - (-) **Cannot be used** when target network **architectures are different** from source model

pre-trained parameters

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

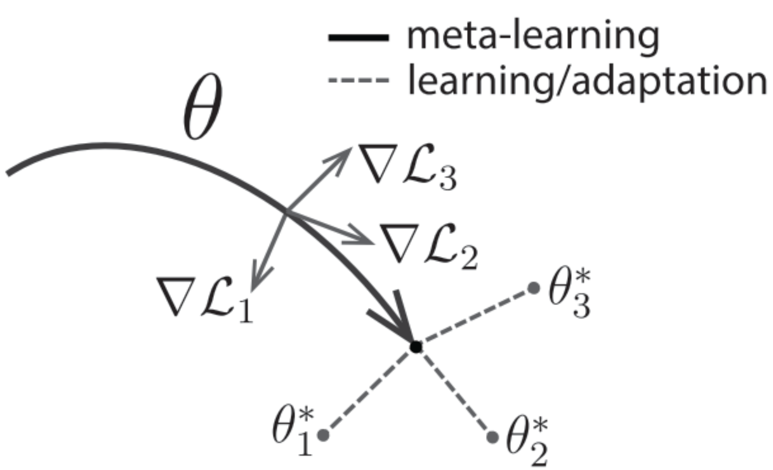
(new) test task

- ***Learning initializations*** of a network that
  - **Adapt fast** with a small number of examples (few-shot learning)
  - Simple and easily generalized to various **model architecture and tasks**

Model-Agnostic Meta-Learning (MAML)

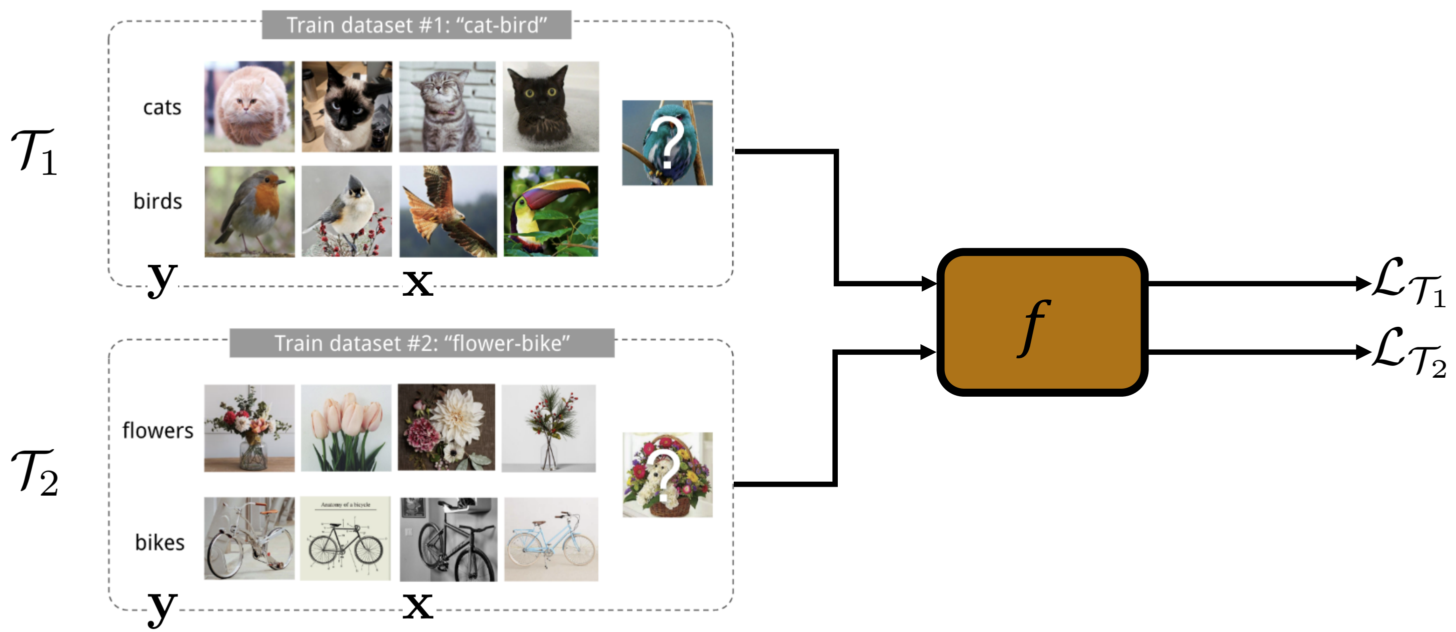
- Key idea
  - Train over many tasks, to learn parameter θ that transfers well
  - Use objective that **encourage** θ to **fast adapt** when fine-tuned with small data
  - Assumption: some representations are more transferrable than others

- Model find parameter θ that would reduce the validation loss on each task
  - To do that, find (one or more steps of) fine-tuned parameter from θ for each task
  - And reduce the validation loss at fine-tuned parameter for each task
  - Meta-update the θ to direction that would adapt faster on each new task

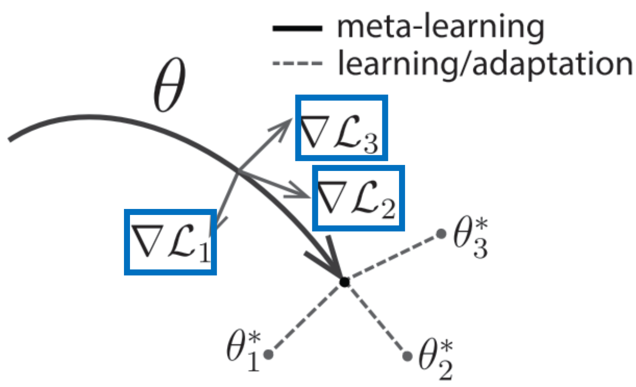


- Notations and problem set-up
  - Task $\mathcal{T} = \{\mathbf{x}, \mathbf{y}, \mathcal{L}(\mathbf{x}, \mathbf{y})\}$
  - Consider a distribution over tasks $p(\mathcal{T})$
  - Model is trained to learn new task $\mathcal{T}_i \sim p(\mathcal{T})$ from only K samples
  - Loss function for task $\mathcal{T}_i$ is $\mathcal{L}_{\mathcal{T}_i}$
  - Model f is learned by minimizing the test error on new samples from $\mathcal{T}_i$

Meta-train set
($K = 4$ samples per class)



- Consider a model f_θ parameterized with θ
- Inner-loop
  - Adapting model to a new task $\mathcal{T}_i$



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

Where α is learning rate,

  - We can compute θ_i' with one or more gradient descent update steps
- Outer-loop
  - Model parameters are trained by optimizing the performance of $f_{\theta_i'}$
  - With respect to θ across tasks sampled from $p(\mathcal{T})$

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

  - So, the meta-optimization:

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

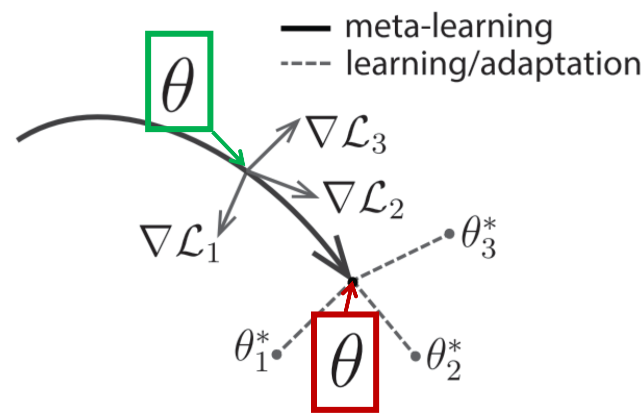
Where β is meta-learning rate

- Consider a model f_θ parameterized with θ
- Inner-loop
  - Adapting model to a new task $\mathcal{T}_i$

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

Where α is learning rate,

  - We can compute θ_i' with one or more gradient d



θ that would adapt better than θ

- Outer-loop
  - Model parameters are trained by optimizing the performance of $f_{\theta_i'}$

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

  - So, the meta-optimization:

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

Where β is meta-learning rate

Meta-Gradients of MAML

- MAML computes 2nd gradients
  - 1-step optimization example

Task-specificly optimized parameters

Meta-learned initial model parameters

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

$$\begin{aligned} g_{\mathrm{MAML}} = \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(\theta') &= (\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})) \cdot (\nabla_{\theta} \theta') \\ &= (\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})) \cdot (\nabla_{\theta}(\theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}))) \end{aligned}$$

  - High computation cost
  - Computation cost is increased with a number of inner-loop iterations T

First Order Approximation of MAML

- MAML computes 2nd gradients
  - 1-step optimization example

Task-specificly optimized parameters

Meta-learned initial model parameters

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

$$\begin{aligned} g_{\mathrm{MAML}} = \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta') &= (\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})) \cdot (\nabla_\theta \theta') \\ &= (\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})) \cdot (\nabla_\theta(\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta))) \end{aligned}$$

  - High computation cost
  - Computation cost is increased with a number of inner-loop iterations T

- Use 1st order approximation

$$\begin{aligned} g_{\mathrm{MAML}} = \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta') &\approx (\nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'})) \cdot (\nabla_\theta \theta) \\ &= \nabla_{\theta'} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'}) \end{aligned}$$

  - Ignore 2nd order terms
  - Empirically show similar performance

MAML

- Inner loop
  - One (or more) step of SGD on training loss starting from a meta-learned network
- Outer loop
  - **Meta-parameters**: initial weights of neural network
  - **Meta-objective** $\mathcal{L}_{\texttt{mo}}$: validation loss
  - **Meta-optimizer**: SGD
- Learned model initial parameters adapt fast to new tasks

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        Evaluate ∇_θ L_{T_i}(f_θ) with respect to K examples
        Compute adapted parameters with gradient descent: θ'_i = θ − α∇_θ L_{T_i}(f_θ)
    end for
    Update θ ← θ − β∇_θ Σ_{T_i~p(T)} L_{T_i}(f_{θ'_i})
end while
```

Inner loop

Outer loop

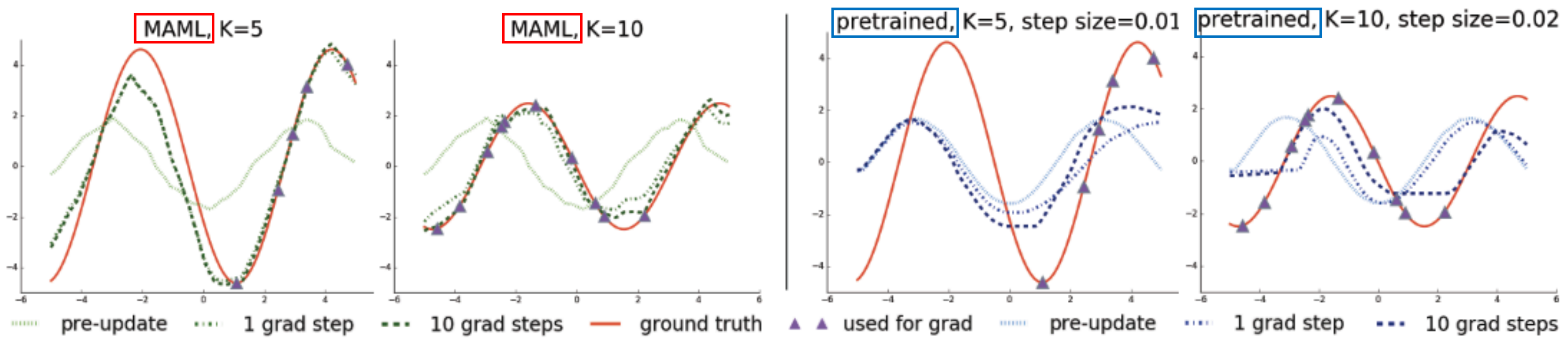
Experiments on Few-Shot Learning Tasks

- Few-shot regression experiments
  - Regress the sine wave $y = A\sin(wx)$
  - Where $A \in [0.1, 5.0]$, $w \in [0, \pi]$, $x \in [-5, 5]$ are randomly sampled
  - MAML with one gradient update inner loop
  - Evaluate performance by fine-tuning the model
    - On K-samples, compared with simply pre-trained model

Experiments on Few-Shot Learning Tasks

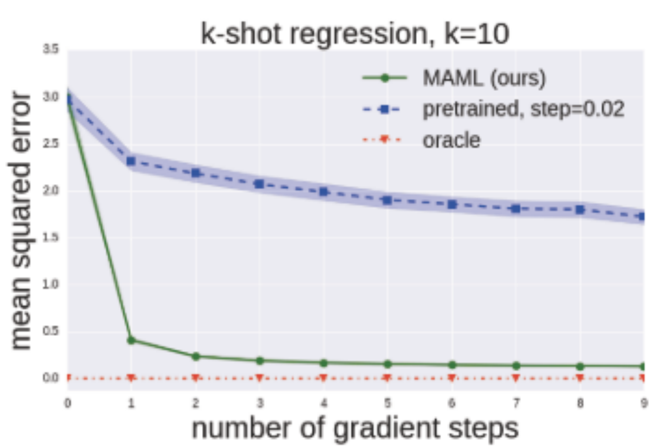
- Few-shot regression experiments
  - Regress the sine wave $y = A\sin(wx)$
  - Where $A \in [0.1, 5.0]$, $w \in [0, \pi]$, $x \in [-5, 5]$ are randomly sampled
  - MAML with one gradient update inner loop
  - Evaluate performance by fine-tuning the model
    - On K-samples, compared with simply pre-trained model

- Adapt much faster with small number of samples (purple triangle below)
  - MAML regresses well in the region without data (learn periodic nature of sine well)



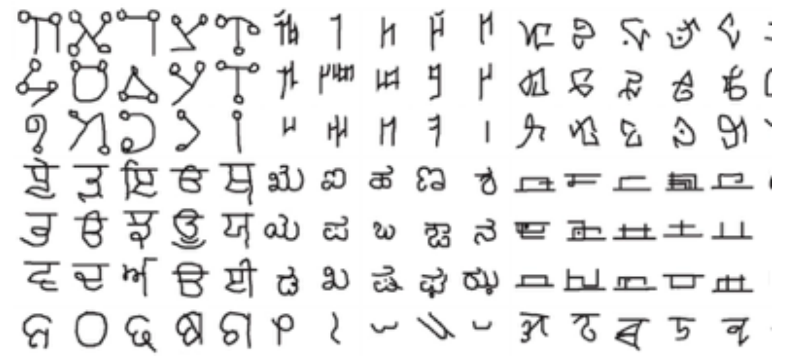
- Few-shot regression experiments
  - Regress the sine wave $y = A\sin(wx)$
  - Where $A \in [0.1, 5.0]$, $w \in [0, \pi]$, $x \in [-5, 5]$ are randomly sampled
  - MAML with one gradient update inner loop
  - Evaluate performance by fine-tuning the model
    - On K-samples, compared with simply pre-trained model

- Adapt much faster with small number of samples (purple triangle below)
  - **Continue to improve** with additional gradient step
    - Not overfitted to θ that only improves after one step
    - Learn initialization that amenable to fast adaptation

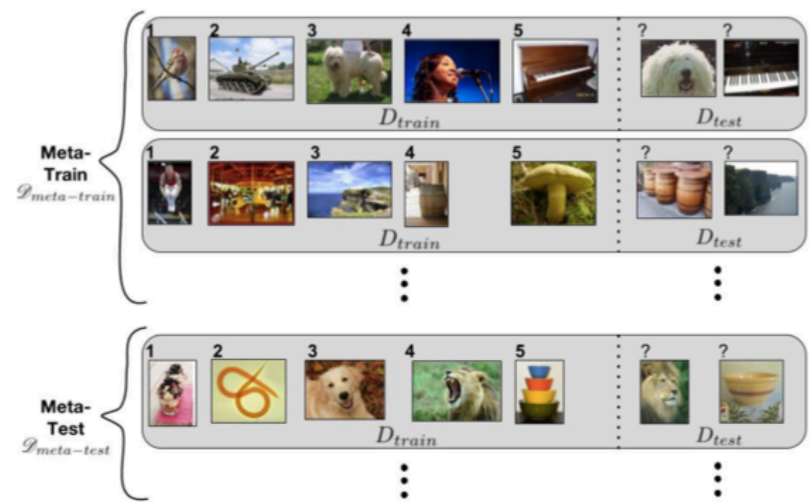


- Datasets for few-shot classification task

- Omniglot
  - Various characters obtained from 50 alphabets
  - Consists of 20 samples of 1623 characters
  - 1200 meta-training, 423 meta-test classes



- Mini-Imagenet
  - Subset of ImageNet
  - 64 training, 12 validation, 24 test classes
  - For each class one/five samples are used



* source : Ravi and Larochelle, Optimization as a model for few-shot learning, ICLR 2017;

Experiments on Few-Shot Learning Tasks

- Few-shot classification experiments
  - Omniglot

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)	$\mathbf{89.7 \pm 1.1\%}$	$\mathbf{97.5 \pm 0.6\%}$	–	–
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)	$\mathbf{98.7 \pm 0.4\%}$	$\mathbf{99.9 \pm 0.1\%}$	$\mathbf{95.8 \pm 0.3\%}$	$\mathbf{98.9 \pm 0.2\%}$

  - Mini-ImageNet

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy 1-shot	5-way Accuracy 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)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML, first order approx. (ours)	$\mathbf{48.07 \pm 1.75\%}$	$\mathbf{63.15 \pm 0.91\%}$
MAML (ours)	$\mathbf{48.70 \pm 1.84\%}$	$\mathbf{63.11 \pm 0.92\%}$

MAML

- MAML outperforms other baselines and generalizes well on unseen tasks

- It is model-agnostic
  - **No dependency** on network architectures
  - **Can be used for another task** not only few-shot learning (e.g., reinforcement learning)
  - Easily applicable to many applications

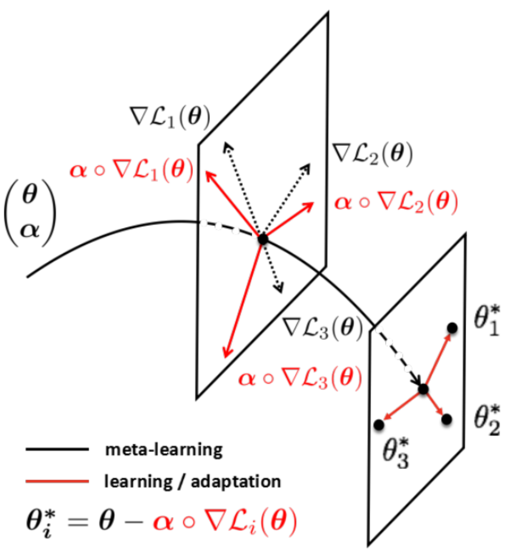
- Many recent works on meta-learning based on MAML
  - Learning the learning rate as well [Li, et. al., 2017]
  - First-order approximation of MAML [Nichol, et. al., 2018]
  - Probabilistic MAML [Finn, et. al., 2018]
  - Visual imitation learning [Finn, et. al., 2017]

An Extension: Meta-SGD - Learning Initialization and Learning Rates

- MAML uses the same learning rate for all the task

- **Meta-SGD** improves MAML by
  - Learning the learning rates for each task
  - Here the learning rates are vector, so that adjust the **gradient direction** as well

- Inner loop computation becomes: $\boldsymbol{\theta}' = \boldsymbol{\theta} - \boldsymbol{\alpha} \circ \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}})$
  - Where $\boldsymbol{\alpha}$ is a vector of learning rates



* source : Li et. al., , Meta-SGD: Learning to Learn Quickly for Few-Shot Learning, 2017;

Experimental Results on Few-Shot Regression

- Same few-shot regression experiment settings with MAML
  - By learning the hyperparameter (learning rates) Meta-SGD outperforms MAML

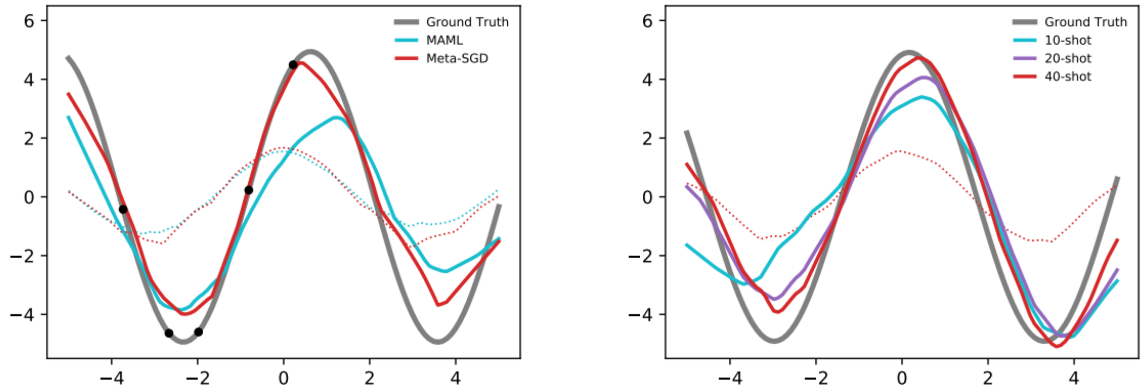


Figure 3: **Left:** Meta-SGD vs MAML on 5-shot regression. Both initialization (dotted) and result after one-step adaptation (solid) are shown. **Right:** Meta-SGD (10-shot meta-training) performs better with more training examples in meta-testing.

Table 1: Meta-SGD vs MAML on few-shot regression

Meta-training	Models	5-shot testing	10-shot testing	20-shot testing
5-shot training	MAML	1.13 ± 0.18	0.85 ± 0.14	0.71 ± 0.12
	Meta-SGD	$\mathbf{0.90 \pm 0.16}$	$\mathbf{0.63 \pm 0.12}$	$\mathbf{0.50 \pm 0.10}$
10-shot training	MAML	1.17 ± 0.16	0.77 ± 0.11	0.56 ± 0.08
	Meta-SGD	$\mathbf{0.88 \pm 0.14}$	$\mathbf{0.53 \pm 0.09}$	$\mathbf{0.35 \pm 0.06}$
20-shot training	MAML	1.29 ± 0.20	0.76 ± 0.12	0.48 ± 0.08
	Meta-SGD	$\mathbf{1.01 \pm 0.17}$	$\mathbf{0.54 \pm 0.08}$	$\mathbf{0.31 \pm 0.05}$

Experimental Results on Few-Shot Classification

- Omniglot experiments

Table 2: Classification accuracies on Omniglot

	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Siamese Nets	97.3%	98.4%	88.2%	97.0%
Matching Nets	98.1%	98.9%	93.8%	98.5%
MAML	$98.7 \pm 0.4\%$	$99.9 \pm 0.1\%$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$
Meta-SGD	$\mathbf{99.53 \pm 0.26}\%$	$\mathbf{99.93 \pm 0.09}\%$	$\mathbf{95.93 \pm 0.38}\%$	$\mathbf{98.97 \pm 0.19}\%$

- Mini-Imagenet experiments

Table 3: Classification accuracies on MiniImagenet

	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Matching Nets	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$	$17.31 \pm 0.22\%$	$22.69 \pm 0.20\%$
Meta-LSTM	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$	$16.70 \pm 0.23\%$	$26.06 \pm 0.25\%$
MAML	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$	$16.49 \pm 0.58\%$	$19.29 \pm 0.29\%$
Meta-SGD	$\mathbf{50.47 \pm 1.87}\%$	$\mathbf{64.03 \pm 0.94}\%$	$\mathbf{17.56 \pm 0.64}\%$	$\mathbf{28.92 \pm 0.35}\%$

- Meta-SGD outperforms baselines with a large margin
    - Especially, it works well with many number of classes (20-way)

- Meta-SGD outperforms MAML in many experiments
  - Learning hyperparameter is useful as well
  - Indicate **simple hyperparameter learning** also gives benefit

- In many meta-learning methods meta-networks learn also:
  - Optimizer parameters: Learning rates, momentum, or optimizer itself
  - Metric space for data distribution similarity comparison
  - Weights of loss for each sample for handling data imbalance
  - And many other *learning rules*

Next, learning optimizers

Table of Contents

1. **Meta-Learning**
   - What is meta-learning?
   - Base learning vs. meta-learning

2. **Types of Meta-Learning**
   - Learning model initialization
   - Learning optimizers

- Learning DNNs is an optimization problem

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta)$$

  - $\mathcal{L}$ be a task-specific objective (e.g., cross-entropy for classification)
  - θ be parameters of a neural network

- How to find the optimal θ^* which minimize $\mathcal{L}$?
  - The parameters are updated iteratively by taking gradient

$$\theta_{t+1} = \theta_t - \gamma \nabla \mathcal{L}(\theta_t)$$

  - DNNs are often trained via **"hand-designed"** gradient-based optimizers
    - e.g., Nesterov momentum [Nesterov, 1983], Adagrad [Duchi et al., 2011], RMSProp [Tieleman and Hinton, 2012], ADAM [Kingma and Ba, 2015]

- Update rules of SGD with momentum:

$$\theta_{t+1} = \theta_t - m_t \qquad\qquad m_t = \mu m_{t-1} + \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

where γ is a learning rate and μ is a momentum

- Unroll the update steps

$$\begin{tikzcd}
\text{Parameters } \theta & \text{Gradients} & \text{Optimizer} & \text{Updates} \\
\boxed{\theta_0} \arrow[r] & \boxed{\nabla_\theta \mathcal{L}(\theta_0)} \arrow[r] & \boxed{m_0 = \gamma \nabla_\theta \mathcal{L}(\theta_0)} \arrow[r] \arrow[d] & \boxed{\Delta\theta_0 = -m_0} \\
\boxed{\theta_1 = \theta_0 + \Delta\theta_0} \arrow[r] & \boxed{\nabla_\theta \mathcal{L}(\theta_1)} \arrow[r] & \boxed{m_1 = \mu m_0 + \gamma \nabla_\theta \mathcal{L}(\theta_1)} \arrow[r] \arrow[d] & \boxed{\Delta\theta_1 = -m_1} \\
\boxed{\theta_2 = \theta_1 + \Delta\theta_1} \arrow[r] & \boxed{\nabla_\theta \mathcal{L}(\theta_2)} \arrow[r] & \boxed{m_2 = \mu m_1 + \gamma \nabla_\theta \mathcal{L}(\theta_2)} \arrow[r] & \boxed{\Delta\theta_2 = -m_2}
\end{tikzcd}$$

- Update rules of ADAM [Kingma and Ba, 2015]:

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{v_t}} m_t \qquad \begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2 \end{aligned}$$

where γ is a learning rate and β_1, β_2 are decay rates for the moments

- Unroll the update steps

Parameters θ	Gradients	Optimizer	Updates
θ_0	$\nabla_\theta \mathcal{L}(\theta_0)$	$m_0 = (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_0)$ $v_0 = (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_0))^2$	$\Delta\theta_0 = -\frac{\gamma}{\sqrt{v_0}} m_0$
$\theta_1 = \theta_0 + \Delta\theta_0$	$\nabla_\theta \mathcal{L}(\theta_1)$	$m_1 = \beta_1 m_0 + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_1)$ $v_1 = \beta_2 v_0 + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_1))^2$	$\Delta\theta_1 = -\frac{\gamma}{\sqrt{v_1}} m_1$
$\theta_2 = \theta_1 + \Delta\theta_1$	$\nabla_\theta \mathcal{L}(\theta_2)$	$m_2 = \beta_1 m_1 + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_2)$ $v_2 = \beta_2 v_1 + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_2))^2$	$\Delta\theta_2 = -\frac{\gamma}{\sqrt{v_2}} m_2$

No Free Lunch Theorem [Wolpert and Macready, 1997]
No algorithm is able to do better than a random strategy in expectation

- **Drawbacks** of these hand-designed optimizers (or update rules)
  - **Potentially poor performance** on some problems
  - **Difficult to hand-craft** the optimizer for **every specific class of functions** to optimize

- **Solution:** Learning an optimizer in an automatic way [Andrychowicz et al., 2016]
  - Explicitly **model optimizers using recurrent neural networks** (RNNs)

$$\theta_{t+1} = \theta_t + \underbrace{g_\phi(\nabla\mathcal{L}(\theta_t), h_t)}_{\text{Outputs of RNN}} \qquad h_t = f_\phi(\underbrace{\nabla\mathcal{L}(\theta_t)}_{\text{Inputs}}, \underbrace{h_{t-1}}_{\text{Hidden states}})$$

  - Cast an optimizer design **as a learning problem**

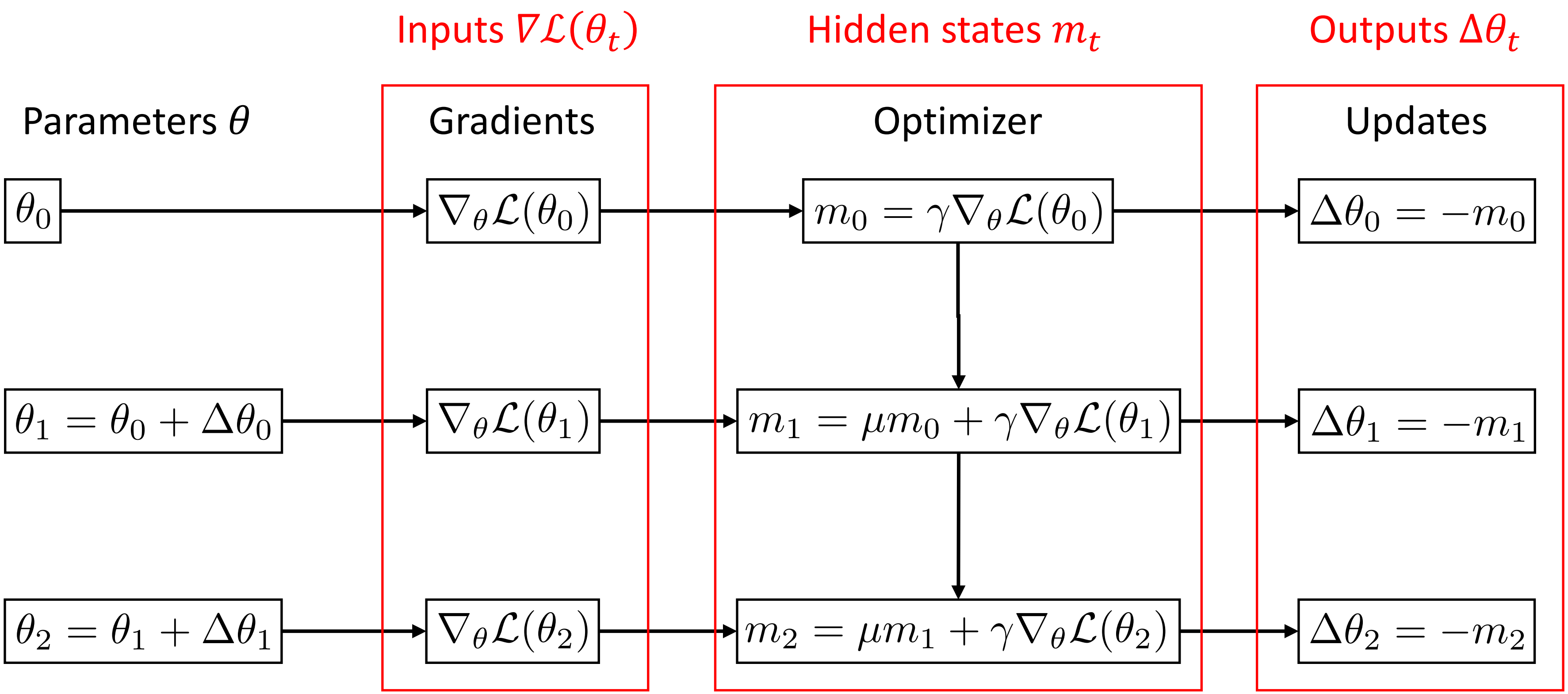
$$\phi^* = \arg\min_{\phi} \mathcal{L}(\theta_T(\phi))$$

where $\theta_T(\phi)$ are the T-step updated parameters given the RNN optimizer ϕ

- Update rules of SGD with momentum:

$$\theta_{t+1} = \theta_t - m_t \qquad\qquad m_t = \mu m_{t-1} + \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

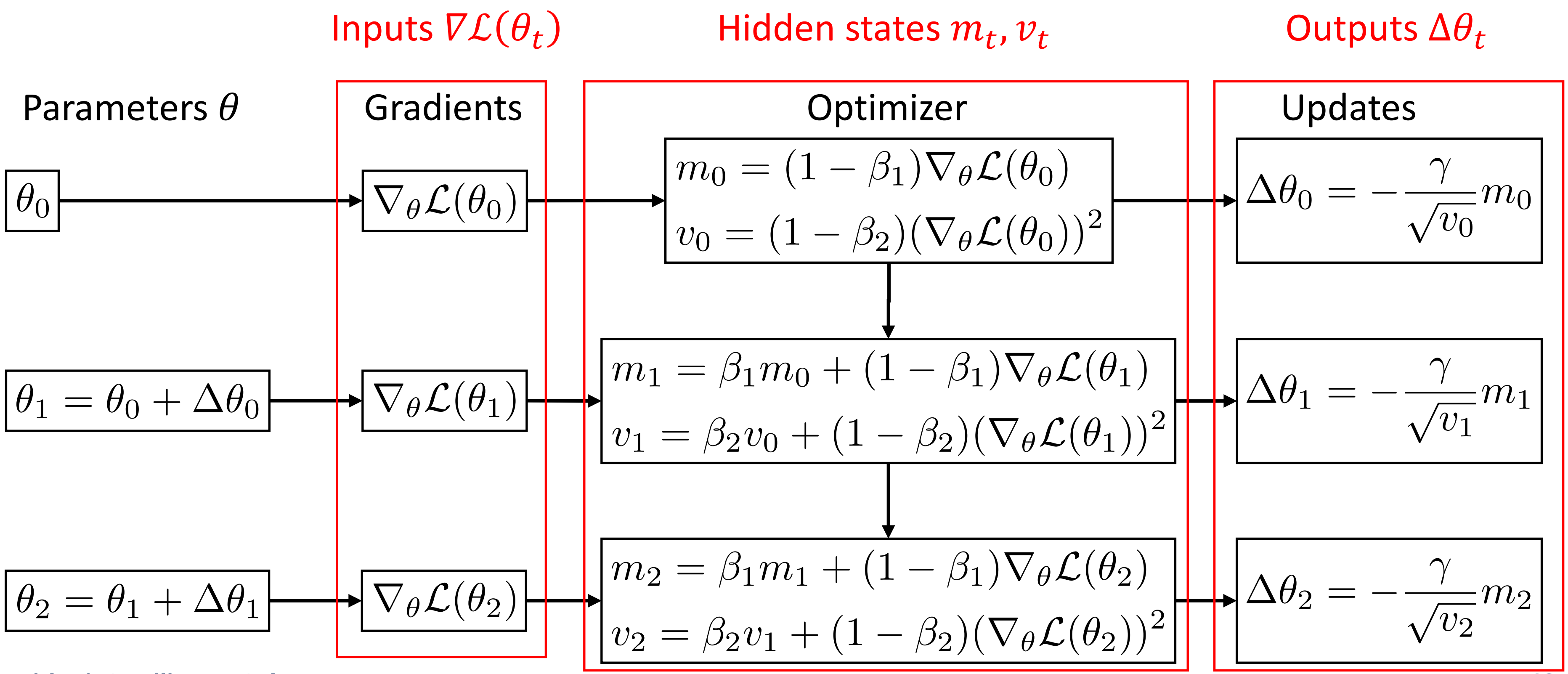
where γ is a learning rate and μ is a momentum



- Update rules of ADAM [Kingma and Ba, 2015]:

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{v_t}} m_t \qquad \begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_t))^2 \end{aligned}$$

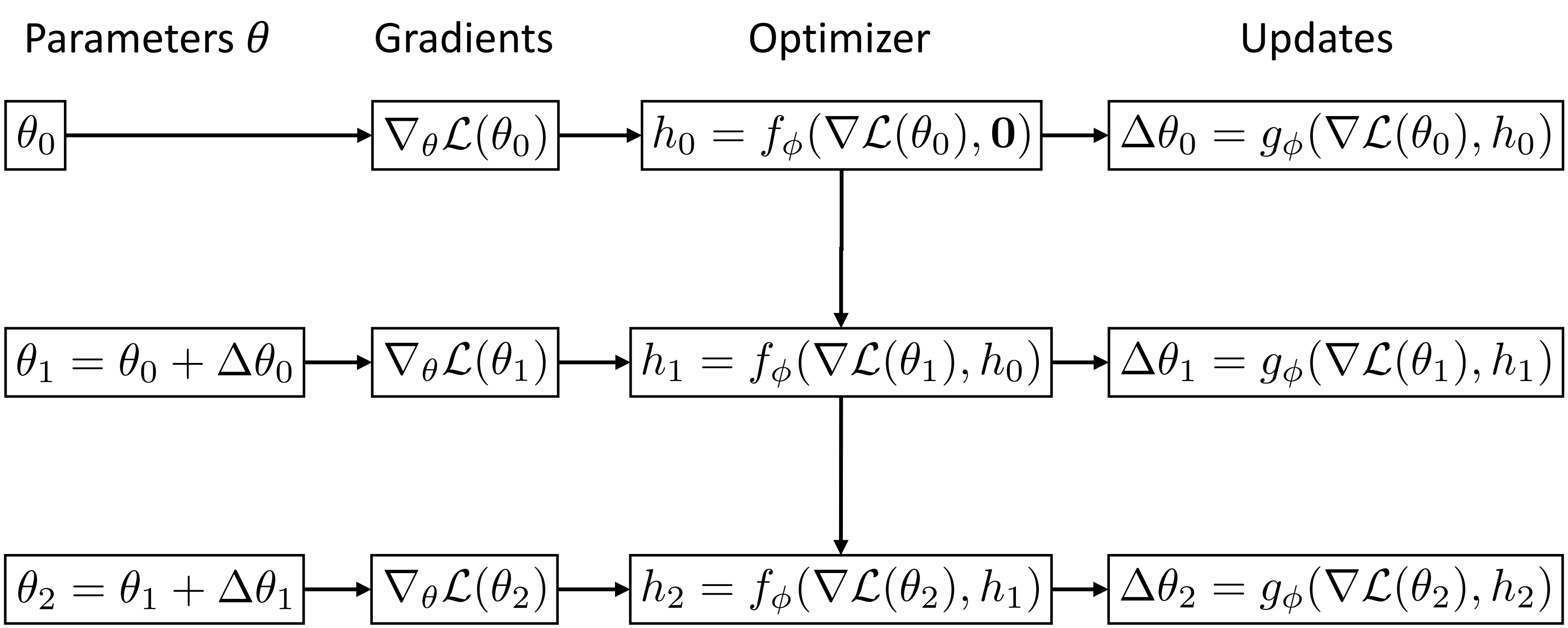
where γ is a learning rate and β_1, β_2 are decay rates for the moments



- Update rules based on a RNN f_ϕ, g_ϕ parameterized by ϕ

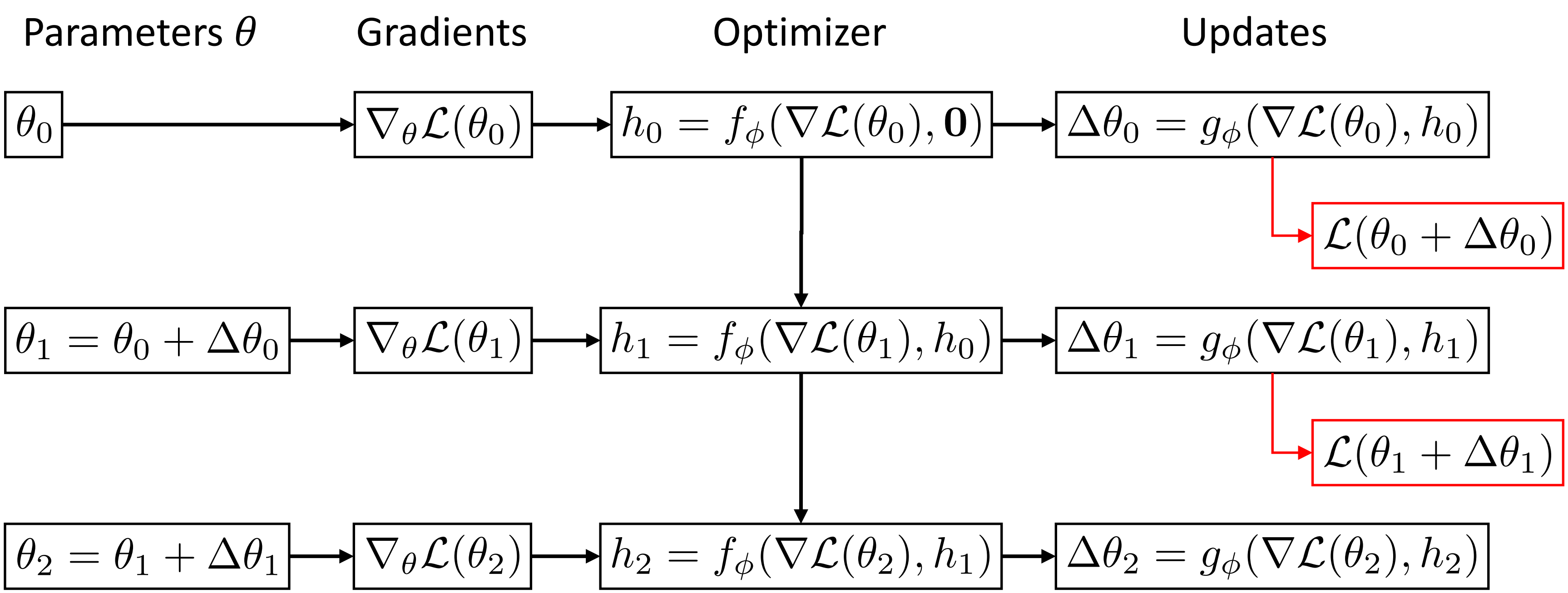
$$\theta_{t+1} = \theta_t + g_\phi(\nabla\mathcal{L}(\theta_t), h_t) \qquad h_t = f_\phi(\nabla\mathcal{L}(\theta_t), h_{t-1})$$

- **Inner-loop:** update the parameters θ via the optimizer for T times



- **Objective for the RNN optimizer ϕ** on the entire training trajectory

$$\mathcal{L}_{\texttt{meta}}(\phi) = \sum_{t=1}^{T} w_t \mathcal{L}(\theta_t) \qquad \text{where } w_t \text{ weights for each time-step}$$



- **Objective for the RNN optimizer ϕ** on the entire training trajectory

$$\mathcal{L}_{\texttt{meta}}(\phi) = \sum_{t=1}^{T} w_t \mathcal{L}(\theta_t)$$

where w_t weights for each time-step

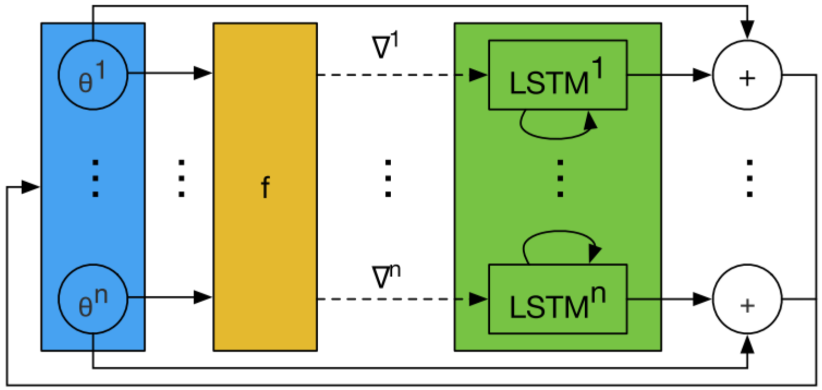
- **Outer-loop:** minimize $\mathcal{L}_{\text{meta}}(\phi)$ **using gradient descent on ϕ**
  - For simplicity, assume $\nabla_\phi \nabla_\theta \mathcal{L}(\theta_t) = 0$ (then, only requires first-order gradients)

Parameters θ	Gradients	Optimizer	Updates
$\boxed{\theta_0}$	$\boxed{\nabla_\theta \mathcal{L}(\theta_0)}$	$\boxed{h_0 = f_\phi(\nabla \mathcal{L}(\theta_0), \mathbf{0})}$	$\boxed{\Delta\theta_0 = g_\phi(\nabla \mathcal{L}(\theta_0), h_0)}$
			$\boxed{\mathcal{L}(\theta_0 + \Delta\theta_0)}$
$\boxed{\theta_1 = \theta_0 + \Delta\theta_0}$	$\boxed{\nabla_\theta \mathcal{L}(\theta_1)}$	$\boxed{h_1 = f_\phi(\nabla \mathcal{L}(\theta_1), h_0)}$	$\boxed{\Delta\theta_1 = g_\phi(\nabla \mathcal{L}(\theta_1), h_1)}$
			backprop $\boxed{\mathcal{L}(\theta_1 + \Delta\theta_1)}$
$\boxed{\theta_2 = \theta_1 + \Delta\theta_1}$	$\boxed{\nabla_\theta \mathcal{L}(\theta_2)}$	$\boxed{h_2 = f_\phi(\nabla \mathcal{L}(\theta_2), h_1)}$	$\boxed{\Delta\theta_2 = g_\phi(\nabla \mathcal{L}(\theta_2), h_2)}$

- A challenge is optimizing (at least) tens of thousands of parameters
  - Computationally not feasible with fully connected RNN architecture

- Use LSTM optimizer which **operates coordinate-wise on the parameters**

- By considering coordinate-wise optimizer
  - Able to use **small network** for optimizer
  - **Share optimizer parameters** across different parameters of the model
    - Input: gradient for single coordinate and the hidden state
    - Output: update for corresponding model parameter



* source : Andrychowicz, et. al., Learning to learn by gradient descent by gradient descent, NIPS 2016

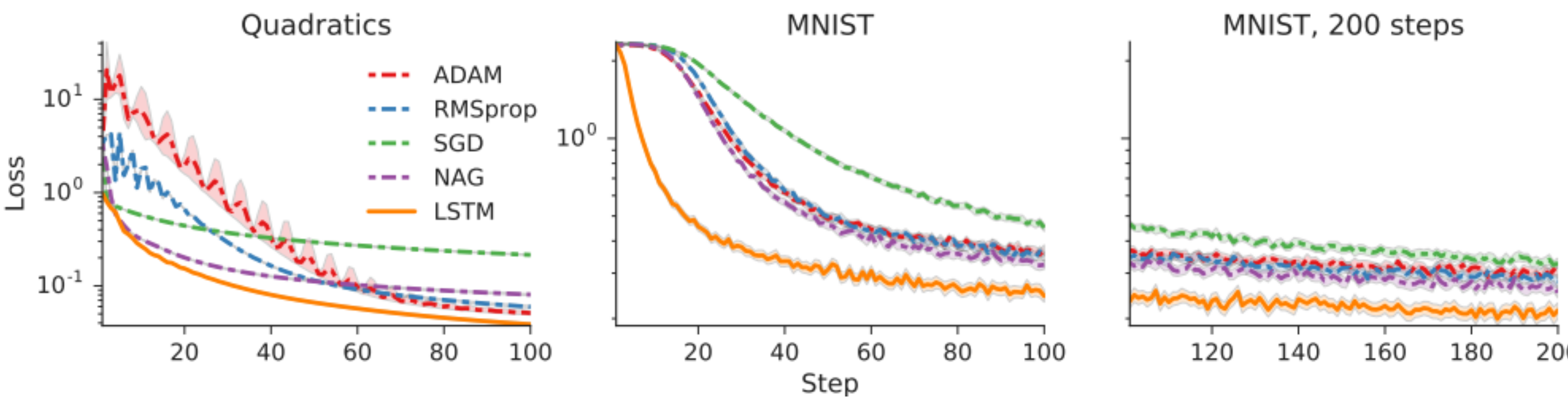
Effectiveness of a Learned Optimizer

- Learning models for
  - Quadratic functions

$$\mathcal{L}(\theta) = \|X\theta - y\|_2^2$$

    - Optimizer is trained by optimizing random functions from this family
    - Tested on newly sampled functions from the same distribution
  - Neural network on MNIST dataset
    - Trained for 100 steps with MLP (1 hidden layer of 20 units, using a sigmoid function)

- Outperform baseline optimizers
  - Also perform well beyond the meta-trained steps (> 100 steps)

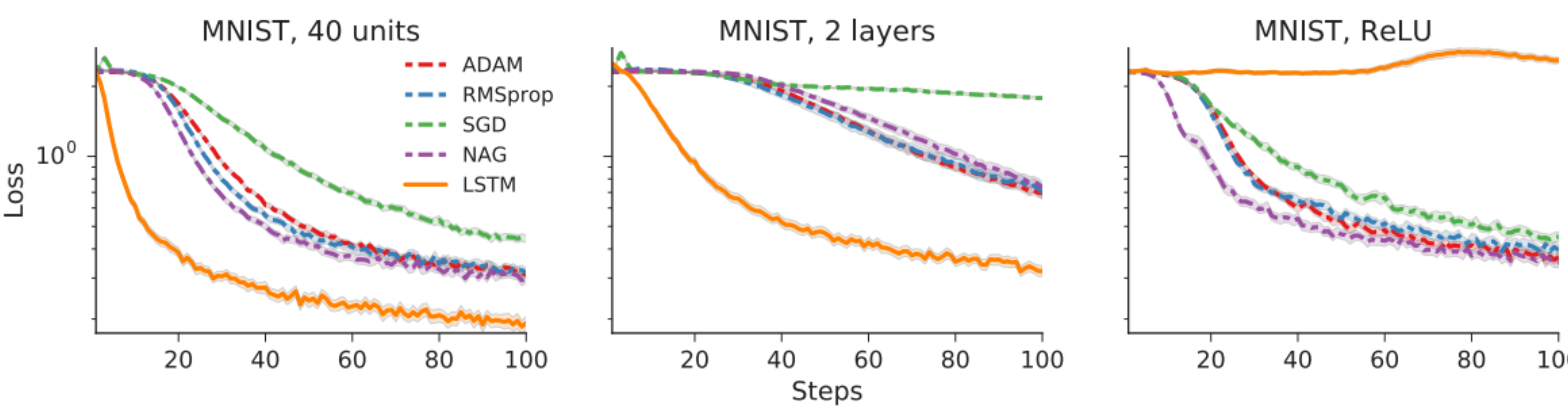


* source : Andrychowicz, et. al., Learning to learn by gradient descent by gradient descent, NIPS 2016

Generalization of a Learned Optimizer

- **Generalization to different architecture models**
  - Learn LSTM optimizer for MNIST dataset
    - With 1 hidden layers (20 units) of sigmoid activation MLP
  - Test generalization ability of a LSTM optimizer for
    - Different **number of hidden units** (20 → 40)
    - Different **number of hidden layers** (1 → 2)
    - Different **activation functions** (Sigmoid → ReLU)

- When learning dynamics are similar, the learned optimizer is generalized well
  - Different activation function significantly changes the problems to solve

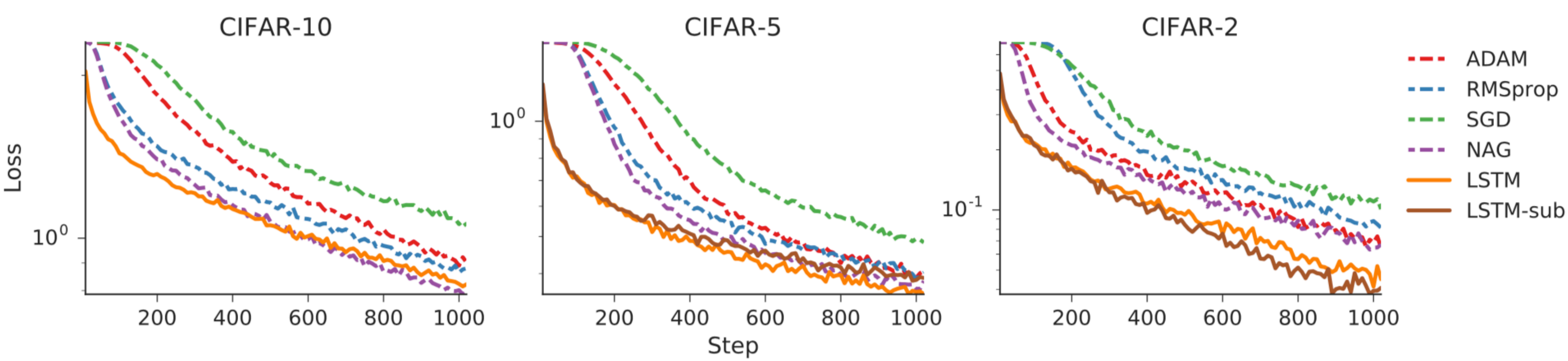


* source : Andrychowicz, et. al., Learning to learn by gradient descent by gradient descent, NIPS 2016

Generalization of a Learned Optimizer

- **Generalization to different datasets**
  - Learn LSTM optimizer on CIFAR-10
  - Test on subset of CIFAR-10 (CIFAR-5 and CIFAR-2)

- Learn much faster than baseline optimizers
  - Even for different (but similar) dataset
  - Without additional tuning of the learned optimizer

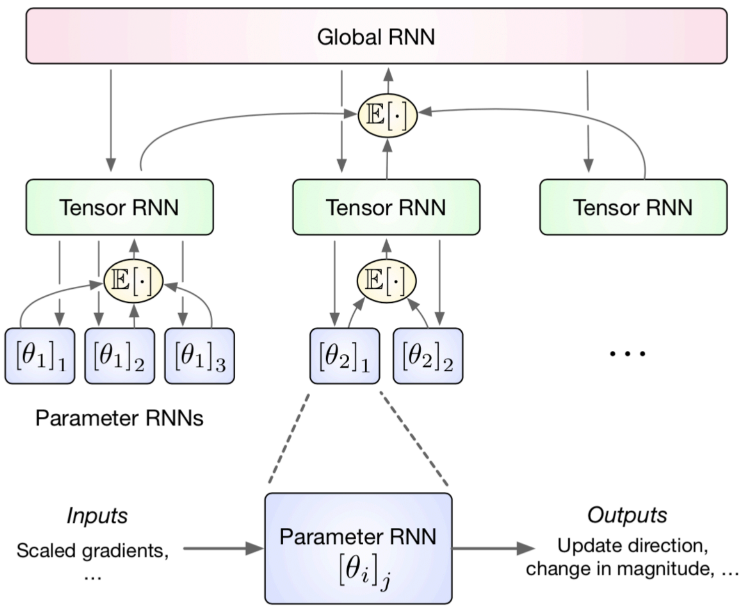


* source : Andrychowicz, et. al., Learning to learn by gradient descent by gradient descent, NIPS 2016

An Extension: Hierarchical RNN Optimizer

- Previous works have have difficulties in:
  - Large problems (e.g., large scale architecture, large number of steps)
  - Generalizing for various tasks

- To tackle these, **hierarchical RNN** is proposed [Wichrowska et al., 2017]



GlobalRNN aggregates global information from all of TensorRNNs

TensorRNN aggregates local information from a subset of ParameterRNNs

TensorRNN and **GlobalRNN** allow ParameterRNNs to have few hidden units

- It generalizes to train Inception/ResNet on ImageNet for thousands of steps

* source : Wichrowska, et. al., Learned Optimizers that Scale and Generalize, ICML 2017

Summary

- Meta-learning is a study about learning the learning rules
  - Make learner perform better without hand-crafted learning rules
- Learning model initialization
  - Learning **initialization** that **transfer well** with small number of samples
- Learning optimizers
  - Optimize the problem **faster and better**
  - In the distribution of the problem that optimizers are meta-trained
- It is applied for many other fields as well
  - Hyperparameter optimization
  - Neural network architecture search
  - Reinforcement learning

References

[Andrychowicz, et. al., 2016] Learning to learn by gradient descent by gradient descent, NIPS 2016
https://arxiv.org/abs/1606.04474

[Vinyals, et. al., 2016] Matching networks for one shot learning, NIPS 2016
https://arxiv.org/abs/1606.04080

[Santoro, et. al., 2016] One-shot learning with memory-augmented neural networks, ICML 2016
https://arxiv.org/abs/1605.06065

[Koch, et. al., 2015] Siamese neural networks for one-shot image recognition, ICML workshop 2015
https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

[Ravi and Larochelle, 2017] Optimization as a model for few-shot learning, ICLR 2017
https://openreview.net/pdf?id=rJY0-Kcll

[Lake, et. al., 2015] Human-level concept learning through probabilistic program induction, Science 2015
http://web.mit.edu/cocosci/Papers/Science-2015-Lake-1332-8.pdf

[Jake Snell, et. al., 2017] Prototypical networks for few-shot learning, NIPS 2017
http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning

[Mishra, et. al., 2018] A simple neural attentive meta-learner, ICLR 2018
https://openreview.net/pdf?id=B1DmUzWAW

[Lemke, et. al., 2015] Metalearning: a survey of trends and technologies, Artificial intelligence review, 2015
https://link.springer.com/content/pdf/10.1007%2Fs10462-013-9406-y.pdf

References

[Vilalta, et. al., 2009] Meta-learning-concepts and techniques. *Data mining and knowledge discovery handbook*. Springer, Boston, MA, 2009. 717-731.
https://link.springer.com/content/pdf/10.1007%2F978-0-387-09823-4.pdf

[Metz, et. al., 2018] Learning unsupervised learning rules, 2018
https://arxiv.org/abs/1804.00222

[Li and Malik, 2017] Learning to optimize, ICLR 2017
https://arxiv.org/pdf/1606.01885.pdf

[Wichrowska, et. al., 2017] Learned optimizers that scale and generalize, ICML 2017
https://arxiv.org/pdf/1703.04813.pdf

[Nichol, et. al., 2018] On first-order meta-learning algorithms, 2018
https://arxiv.org/abs/1803.02999

[Finn, et. al., 2017] Model-agnostic meta-learning, ICML 2017
https://arxiv.org/abs/1703.03400

[Finn, et. al., 2018] Probabilistic model-agnostic meta-learning, NIPS 2018
https://arxiv.org/abs/1806.02817

[Finn, et. al., 2017] One-Shot Visual Imitation Learning via Meta-Learning, CoRL 2017
https://arxiv.org/abs/1709.04905

[Metz, et. al., 2018] Learned optimizers that outperform SGD on wall-clock and test loss, 2018
https://arxiv.org/abs/1810.10180

[Li, et. al., 2017] Meta-SGD: Learning to learn quickly for few-shot leanring
https://arxiv.org/pdf/1707.09835.pdf

References

[Nesterov, 1983] A method of solving a convex programming problem with convergence rate o(1/k2), Soviet Mathematics Doklady, 1983

[Duchi et al., 2011] Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011
http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

[Tieleman and Hinton, 2012] Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning, 2012
https://www.coursera.org/learn/machine-learning

[Kingma and Ba, 2015] Adam: A method for stochastic optimization, ICLR 2015
https://arxiv.org/pdf/1412.6980.pdf

[Wolpert and Macready, 1997] No free lunch theorems for optimization, Transactions on Evolutionary Computation, 1997
https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

- *Meta-learning* differs from *base-learning* in the scope of the level of adaptation
  - Instead of focusing on learning a specific task, learn the learning rule
    - Considering **dataset as a data sample** $\mathcal{D}_{\texttt{meta}} = \{D^i_{\texttt{train}}, D^i_{\texttt{test}}\}_{i=1,\cdots,N}$
    - **Learn** *patterns* **across tasks** $\mathcal{T}$
    - Consider distribution of tasks $p(\mathcal{T})$
      - Learning to learn that works well on a task from the distribution
      - **Generalization for new tasks** (not only new data *samples*) from the same distribution

  - Examples
    - **Learning optimizer** itself that works well for specific class of problems
      - Instead of using hand-crafted optimizer (e.g., SGD)
    - **Learning *metric*** that works well for the purpose of a comparison
      - Instead of using some hand-designed *metric* to compare two samples
    - **Learning initializations** that is effective on a specific task (e.g., few-shot learning)
      - Instead of pre-defined model initialization (e.g., pre-trained weights on ImageNet)
    - Detail algorithms of those examples are in later slides

Model-Agnostic Meta-Learning (MAML)

- Key idea
  - **Train over many tasks, to learn parameter θ that transfers**
  - Use objective that encourage θ to be fast adapt when fine-tuned with small data
  - Assumption: some representations are more transferrable than others

- Problem set-up for few-shot learning
  - Task $\mathcal{T} = \{\mathbf{x}, \mathbf{y}, \mathcal{L}(\mathbf{x}, \mathbf{y})\}$
  - During meta-train
    - Consider a distribution over tasks $p(\mathcal{T})$
    - Model is trained to learn new task $\mathcal{T}_i \sim p(\mathcal{T})$ from only K samples
    - Loss function for task $\mathcal{T}_i$ is $\mathcal{L}_{\mathcal{T}_i}$
    - Model f is learned by considering how the test error on new samples from $\mathcal{T}_i$
    - The test error of f is used as the training error of the meta-learning
  - During meta-test
    - New tasks are sampled from $p(\mathcal{T})$
    - Model f is trained for new task with K samples
    - Measure model's performance (i.e., measure meta-performance)