

# Meta learning

**AI602: Recent Advances in Deep Learning**  
**Lecture 10**

**Slide made by**

**Junsu Kim, Changyeon Kim, Jongjin Park**  
**KAIST Graduate School of AI**

### 1. Introduction

- Problem setup
- Overview of meta-learning approaches

### 2. Model-based methods

- Meta-learning with External Memory
- Simple Neural Attentive meta-Learner (SNAIL)

### 3. Metric-based methods

- Matching Networks
- Prototypical Networks
- Relation Networks

### 4. Optimization-based methods

- Training Meta-learner Optimizers
- Model Agnostic Meta-learning (MAML)

## 1. Introduction

- Problem setup
- Overview of meta-learning approaches

## 2. Model-based methods

- Meta-learning with External Memory
- Simple Neural Attentive meta-Learner (SNAIL)

## 3. Metric-based methods

- Matching Networks
- Prototypical Networks
- Relation Networks

## 4. Optimization-based methods

- Training Meta-learner Optimizers
- Model Agnostic Meta-learning (MAML)

# Introduction: What is Few-shot Learning?

- 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 [\[hide\]](#)

- 1 Taxonomy
- 2 Description
- 3 Distribution and habitat
- 4 Behavior
  - 4.1 Sociality
  - 4.2 Breeding





## Introduction: What is Few-shot Learning?

- Which is *American goldfinch*?



## Introduction: What is Few-shot 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

## What is Meta-learning?

---

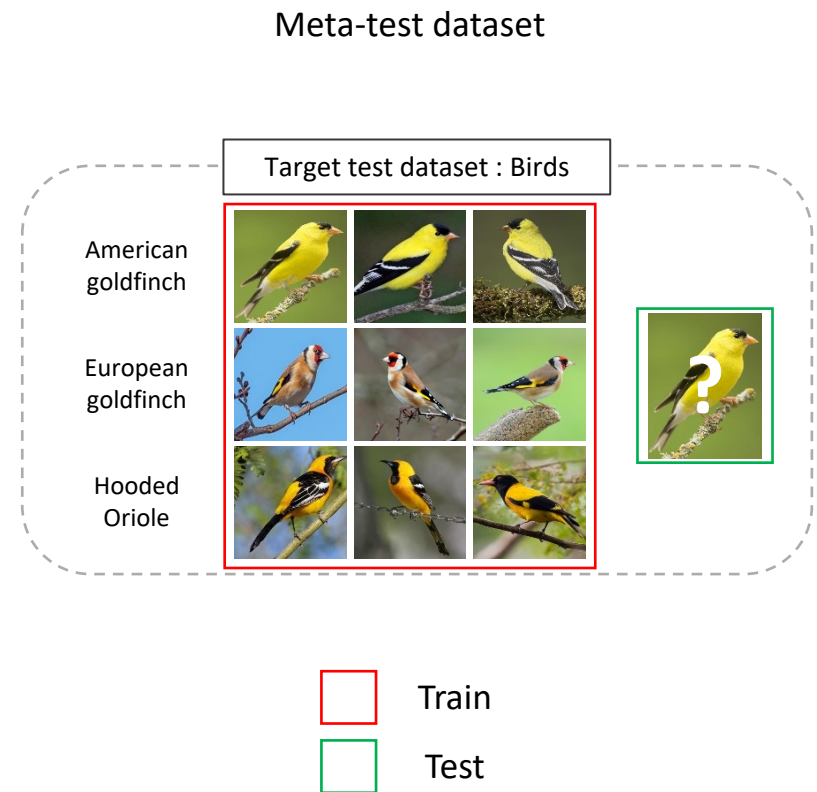
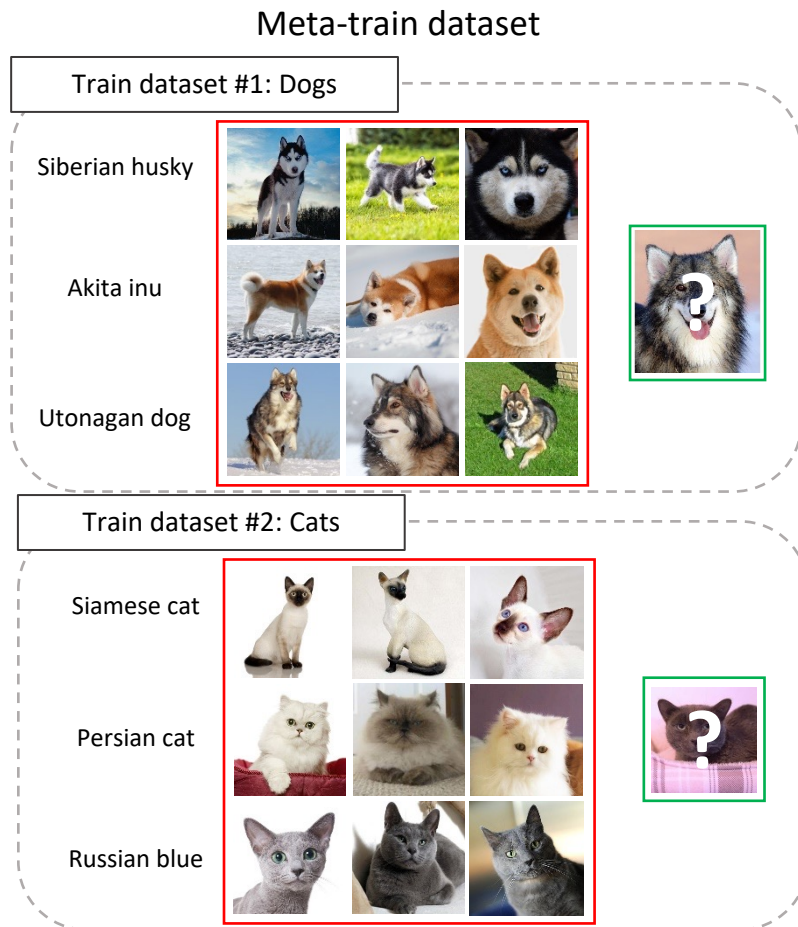
- Definition from Wikipedia:

**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

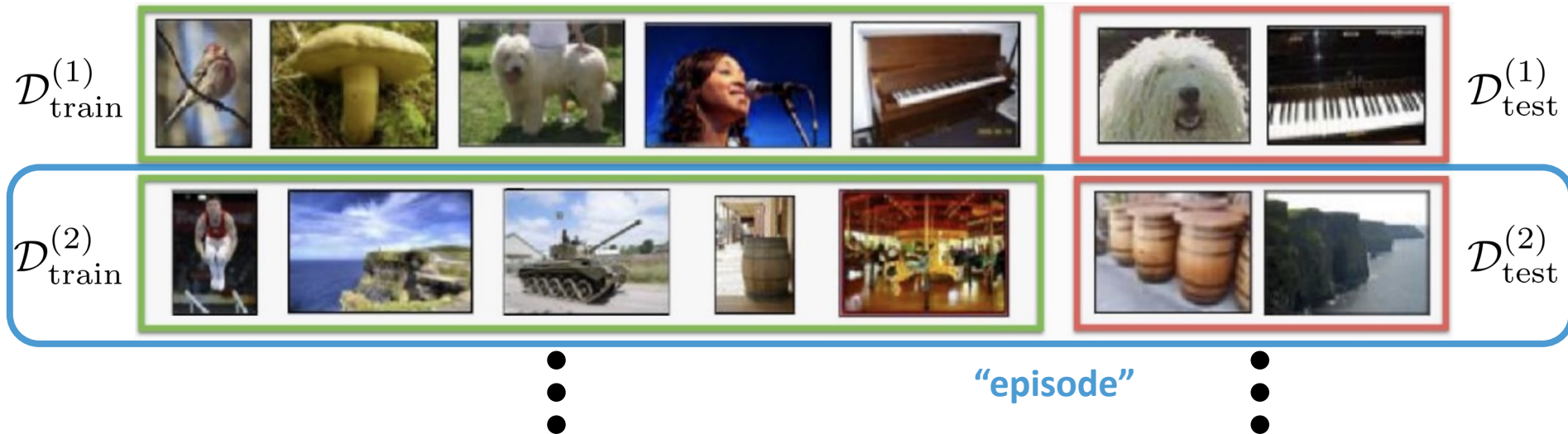
# What is Meta-learning?

- 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



## Formulation of Meta-learning

- In contrast to regular deep learning whose a single training instance is a labeled sample, that for meta-learning is a task episode (e.g., a *set of samples*).

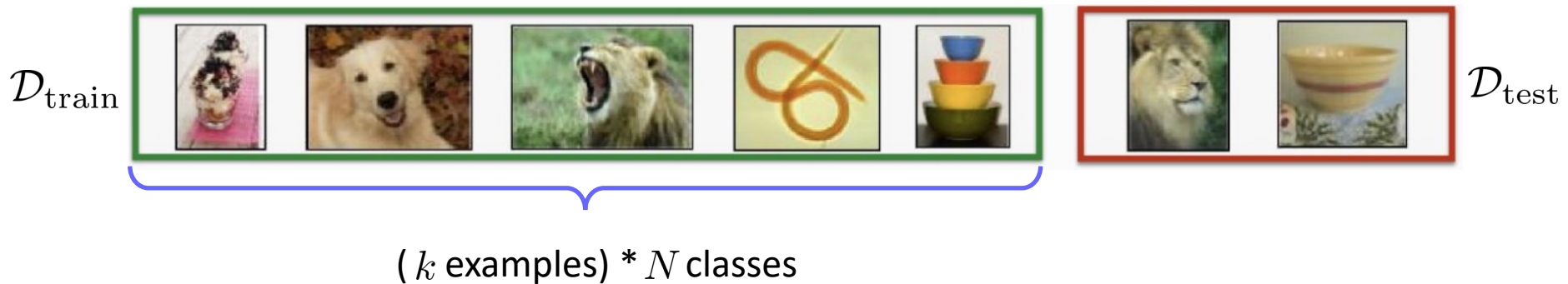


- First we need a training set  $\mathcal{D}_{\text{train}}$  (a.k.a. *support set*) which consists of a couple of samples belonging to a subset of classes  $L$  (or tasks  $\mathcal{T} \sim p(\mathcal{T})$ ).
- We also need to specify samples that we would like to evaluate; then form a test set  $\mathcal{D}_{\text{test}}$  (a.k.a. *query*) belonging to  $L$ .
- Together,  $\mathcal{D}_{\text{train}}$  and  $\mathcal{D}_{\text{test}}$  form a training *episode*. The model takes many episodes one by one.



## Formulation of Meta-learning

- $N$ -way,  $k$ -shot learning task
  - $N$  classes per episode, that have not previously been trained upon.
  - $k$  labelled examples per class



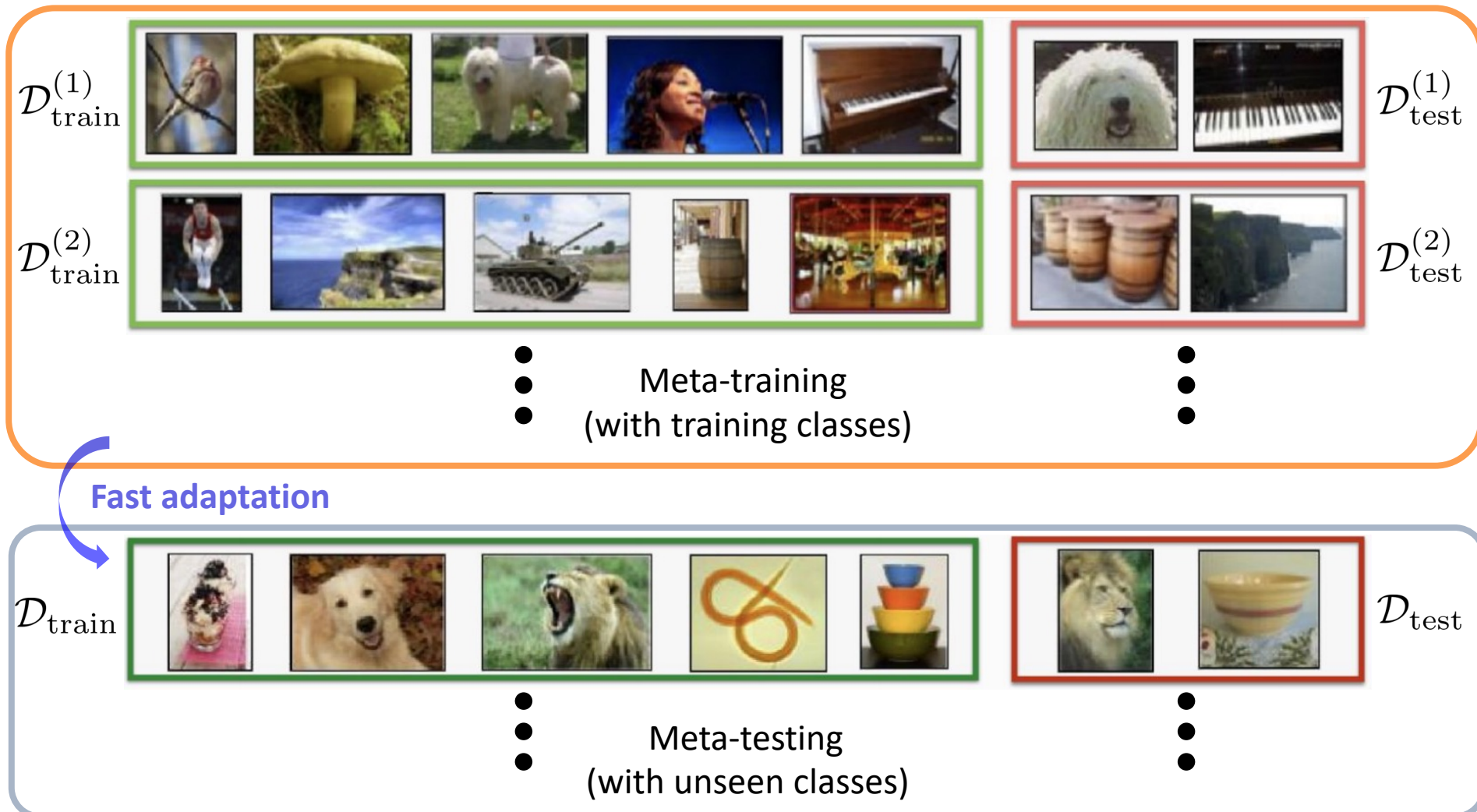
- In classification tasks, the objective of meta-learning is:

$$\theta = \operatorname{argmax}_{\theta} \mathbb{E}_{L \sim \mathcal{T}} \left[ \mathbb{E}_{\mathcal{D}_{\text{train}} \sim L, \mathcal{D}_{\text{test}} \sim L} \left[ \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \overset{\text{log-likelihood}}{\log P_{\theta}(y|x, \mathcal{D}_{\text{train}})} \right] \right]$$

- This can be replaced with any ML problems (e.g., regression, language generation, skill learning)
- We will focus on classification tasks in meta-learning (i.e., *few-shot classification*)

## Formulation of Meta-learning

- In test time, the model must be fast adapted to accommodate new classes not seen in training, given only a few examples of each of the classes.



- Recall the objective of few-shot classification:

$$\theta = \operatorname{argmax}_{\theta} \mathbb{E}_{L \sim \mathcal{T}} \left[ \mathbb{E}_{\mathcal{D}_{\text{train}} \sim L, \mathcal{D}_{\text{test}} \sim L} \left[ \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \log P_{\theta}(y|x, \mathcal{D}_{\text{train}}) \right] \right]$$

How to compute?



- Recall the objective of few-shot classification:

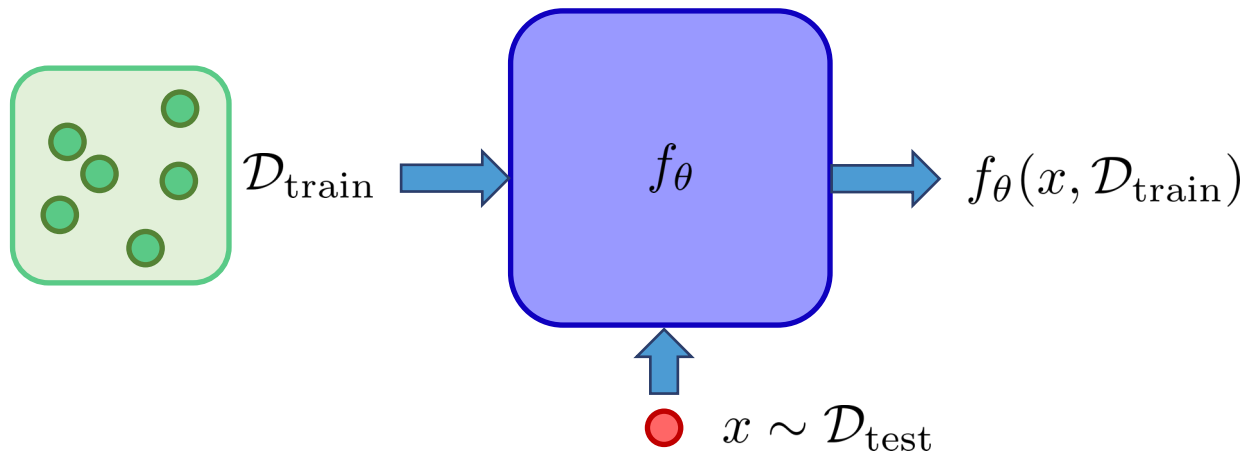
$$\theta = \operatorname{argmax}_{\theta} \mathbb{E}_{L \sim \mathcal{T}} \left[ \mathbb{E}_{\mathcal{D}_{\text{train}} \sim L, \mathcal{D}_{\text{test}} \sim L} \left[ \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \log P_{\theta}(y|x, \mathcal{D}_{\text{train}}) \right] \right]$$

How to compute?

- Model-based Meta-learning**

- The key idea is to build a model and training process designed for rapid generalization across tasks.

$$P_{\theta}(y|x, \mathcal{D}_{\text{train}}) = f_{\theta}(x, \mathcal{D}_{\text{train}})$$



- To compute the set representation of  $\mathcal{D}_{\text{train}}$ , **RNN-based models** are widely utilized.

## Types of Meta-learning

- Recall the objective of few-shot classification:

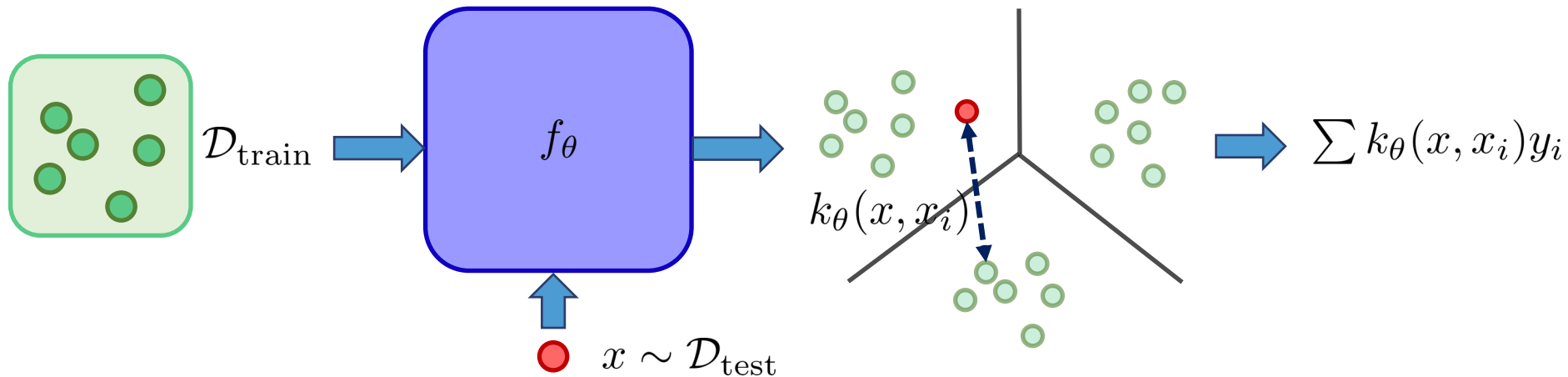
$$\theta = \operatorname{argmax}_{\theta} \mathbb{E}_{L \sim \mathcal{T}} \left[ \mathbb{E}_{\mathcal{D}_{\text{train}} \sim L, \mathcal{D}_{\text{test}} \sim L} \left[ \sum_{(x, y) \in \mathcal{D}_{\text{test}}} \boxed{\log P_{\theta}(y|x, \mathcal{D}_{\text{train}})} \right] \right]$$

How to compute?

- Metric-based Meta-learning**

- The key idea is to learn a metric or distance function on deep neural features over objects.

$$P_{\theta}(y|x, \mathcal{D}_{\text{train}}) = \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} k_{\theta}(x, x_i) y_i$$



- Recall the objective of few-shot classification:

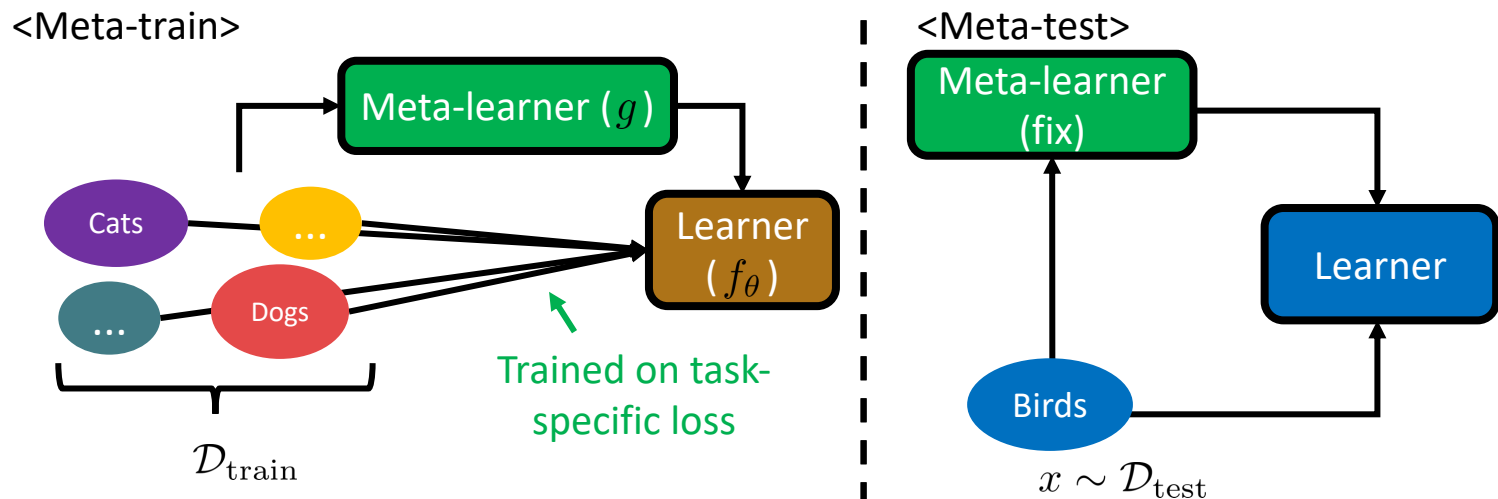
$$\theta = \operatorname{argmax}_{\theta} \mathbb{E}_{L \sim \mathcal{T}} \left[ \mathbb{E}_{\mathcal{D}_{\text{train}} \sim L, \mathcal{D}_{\text{test}} \sim L} \left[ \sum_{(x,y) \in \mathcal{D}_{\text{test}}} \log P_{\theta}(y|x, \mathcal{D}_{\text{train}}) \right] \right]$$

How to compute?

- Optimization-based Meta-learning**

- The key idea is to adjust the optimization algorithm with a few examples.

$$P_{\theta}(y|x, \mathcal{D}_{\text{train}}) = f_{\theta(\mathcal{D}_{\text{train}})}(x)$$



- Gradient-based optimization techniques** (e.g. gradient descent) are utilized.

$$\theta(\mathcal{D}_{\text{train}}) = g\left(\theta_0, \left\{ \nabla_{\theta_0} \mathcal{L}(x_i, y_i) \right\}_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} \right)$$

## 1. Introduction

- Problem setup
- Overview of meta-learning approaches

## 2. **Model-based methods**

- Meta-learning with External Memory
- Simple Neural Attentive meta-Learner (SNAIL)

## 3. Metric-based methods

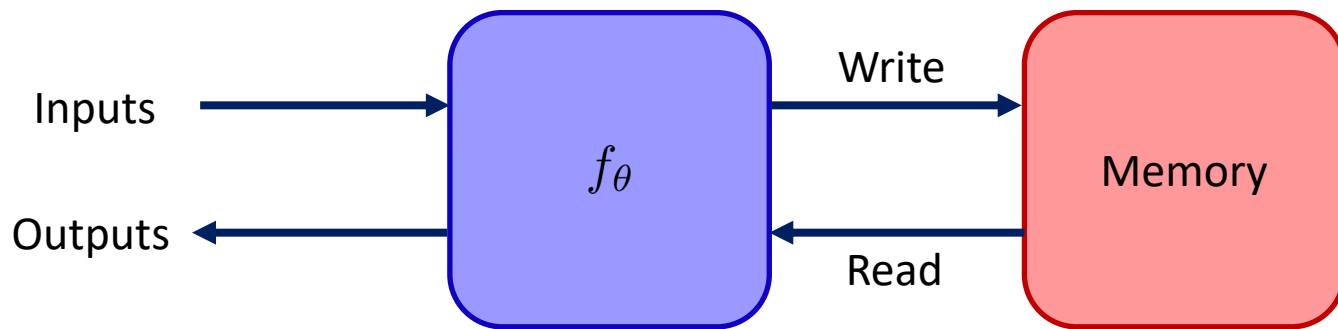
- Matching Networks
- Prototypical Networks
- Relation Networks

## 4. Optimization-based methods

- Training Meta-learner Optimizers
- Model Agnostic Meta-learning (MAML)

## Meta-learning with External Memory

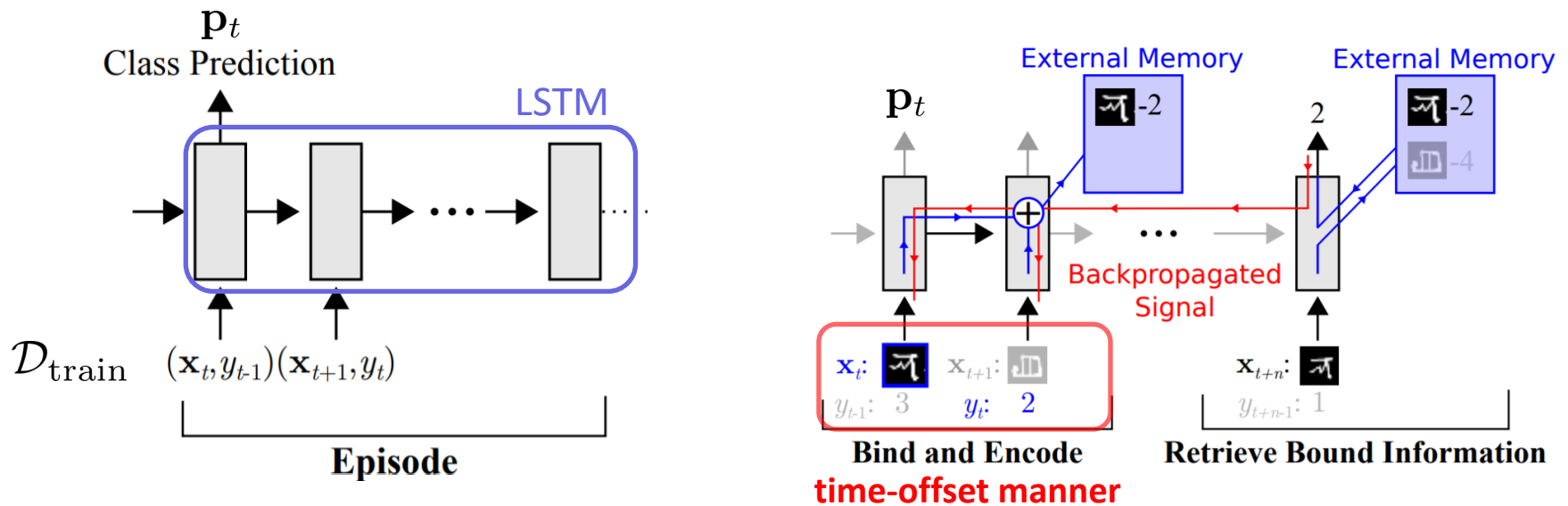
- [Graves et al. 14] propose a Neural Turing Machine (NTM), a neural networks architecture which has external memory.
- With an explicit storage buffer, it is easier for the network to rapidly incorporate new information and not to forget in the future.



- Read and write heads in a NTM external memory module are fully differentiable and trained for rapid memory encoding and retrieval.
- There are some recent works on meta-learning using external memory units.
  - Memory-Augmented Neural Network (MANN) [Santoro et al. 16]
  - Meta Networks [Munkhdalai et. al., 17]

## Meta-learning with External Memory

- [Santoro et al. 16] proposed Memory-Augmented Neural Network (MANN) to rapidly assimilate new data, and to make accurate predictions with few samples.
- They train MANN to perform classification while presenting the data instance and labels in a **time-offset manner** to prevent simple mapping from label to label.

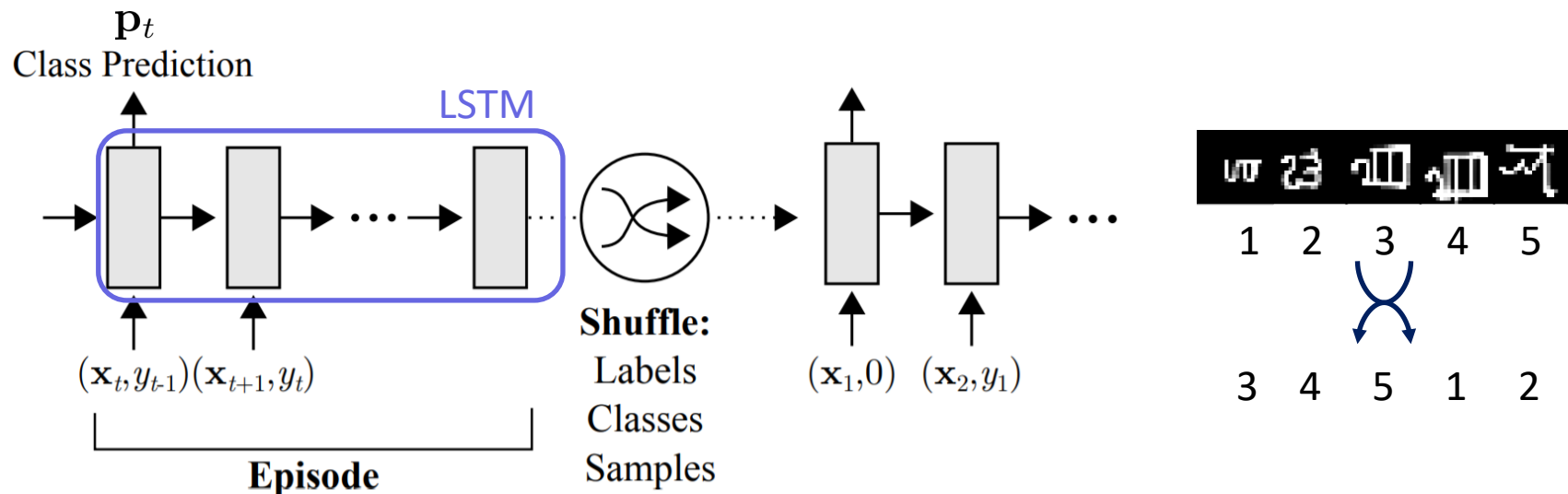


- The loss function is the sum of classification loss at each time step.

$$\mathcal{L}(\theta) = - \sum_t \mathbf{y}_t^T \log \mathbf{p}_t$$

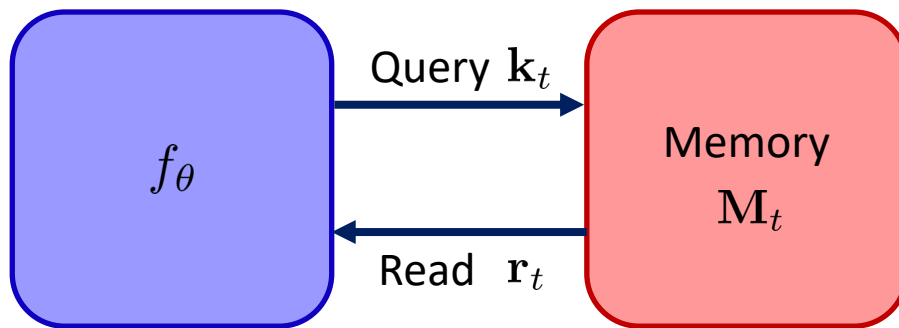
## Meta-learning with External Memory

- [Santoro et al. 16] proposed Memory-Augmented Neural Network (MANN) to rapidly assimilate new data, and to make accurate predictions with few samples.
- They train MANN to perform classification while presenting the data instance and labels in a **time-offset manner** to prevent simple mapping from label to label.



- Further, **they shuffle labels**, classes, and samples from episode to episode to prevent the network from learning sample-class bindings in its weights.

- The network has an external memory module,  $\mathbf{M}_t$ , that is both read from and written to.
- The rows of  $\mathbf{M}_t$  serve as memory 'slots', with the row vectors themselves constituting individual memories.
- For reading, a cosine distance measure is computed for the query key vector (here notated as  $\mathbf{k}_t$ ) and each individual row in memory.



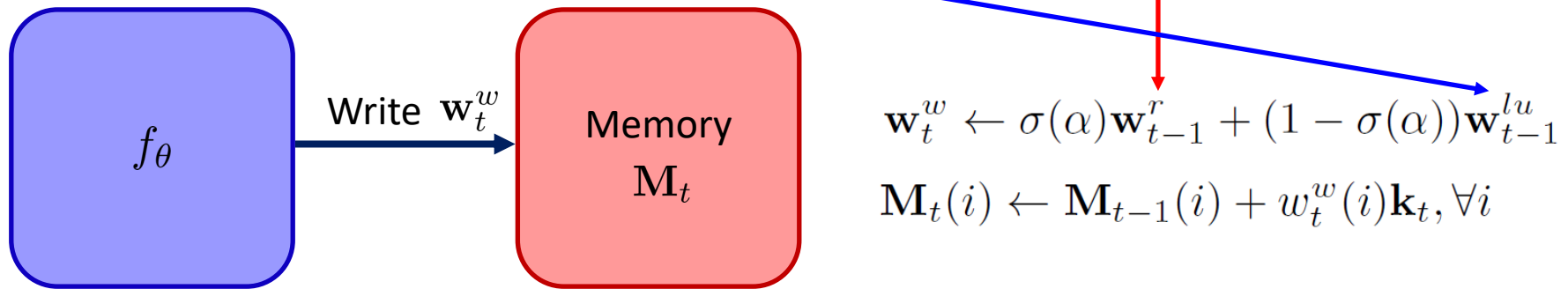
$$K(\mathbf{k}_t, \mathbf{M}_t(i)) = \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\|\mathbf{k}_t\| \|\mathbf{M}_t(i)\|}$$

$$w_t^r(i) \leftarrow \frac{\exp(K(\mathbf{k}_t, \mathbf{M}_t(i)))}{\sum_j \exp(K(\mathbf{k}_t, \mathbf{M}_t(j)))}$$

- A memory,  $\mathbf{r}_t$ , is then retrieved using these read-weights:  $\mathbf{r}_t \leftarrow \sum_i w_t^r(i) \mathbf{M}_t(i)$
- Finally,  $\mathbf{r}_t$  is concatenated with the hidden state of the LSTM model.



- To write to memory, they implemented an access module called Least Recently Used Access (LRUA) which writes to either **the most recently read location**, or **the least recently used location**.



- Usage weights  $\mathbf{w}_t^u$  are computed each time-step to keep track of the locations most recently read or written to:  $\mathbf{w}_t^u \leftarrow \gamma\mathbf{w}_{t-1}^u + \mathbf{w}_t^r + \mathbf{w}_t^w$
- The least-used weights,  $\mathbf{w}_t^{lu}$  is:

$$w_t^{lu}(i) = \begin{cases} 0 & \text{if } w_t^u(i) > m(\mathbf{w}_t^u, n) \\ 1 & \text{if } w_t^u(i) \leq m(\mathbf{w}_t^u, n) \end{cases}$$

where  $m(\mathbf{v}, n)$  denotes the  $n$ -th smallest element of the vector  $\mathbf{v}$ .

## Simple Neural Attentive meta-Learner (SNAIL)

---

- Traditional RNN architectures propagate information by keeping it in their hidden state from one time step to the next.
  - This *temporally-linear dependency* bottlenecks their capacity.
- [Mishra et al. 18] propose a model architectures that addresses this shortcoming.
- They combine these two modules for simple neural attentive learner (SNAIL):
  - **Temporal convolutions**, which enable the meta-learner to aggregate contextual information from past experience
  - **Causal attention**, which allow it to pinpoint specific pieces of information within that context.
- These two components complement each other: while the former provide **high-bandwidth access** at the expense of **finite context size**, the latter provide **pinpoint access** over **an infinitely large context**.

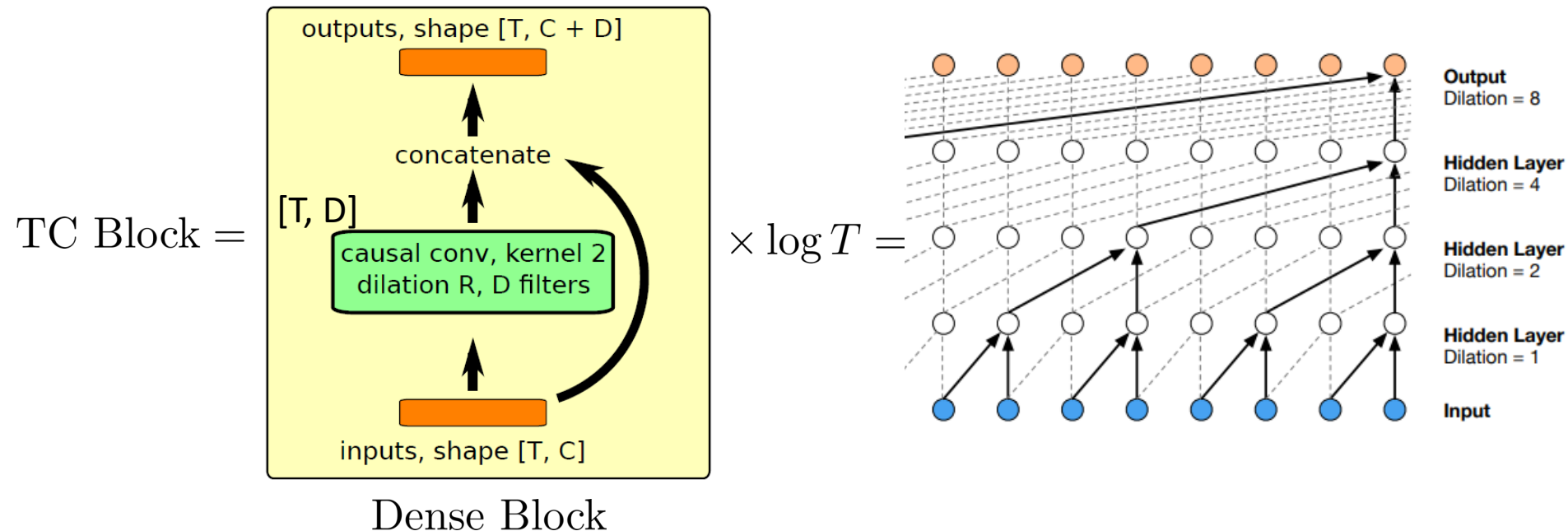
## Simple Neural Attentive meta-Learner (SNAIL)

- Two of the building blocks that compose SNAIL architectures.
- A Dense block applies a causal 1D-convolution, and then concatenates the output to its input. A **Temporal Convolution (TC) block** applies a series of dense blocks with exponentially-increasing dilation rates.

---

```
1: function TCBLOCK(inputs, sequence length  $T$ , number of filters  $D$ ):  
2:   for  $i$  in  $1, \dots, \lceil \log_2 T \rceil$  do  
3:     inputs = DenseBlock(inputs,  $2^i, D$ )  
4:   return inputs
```

---

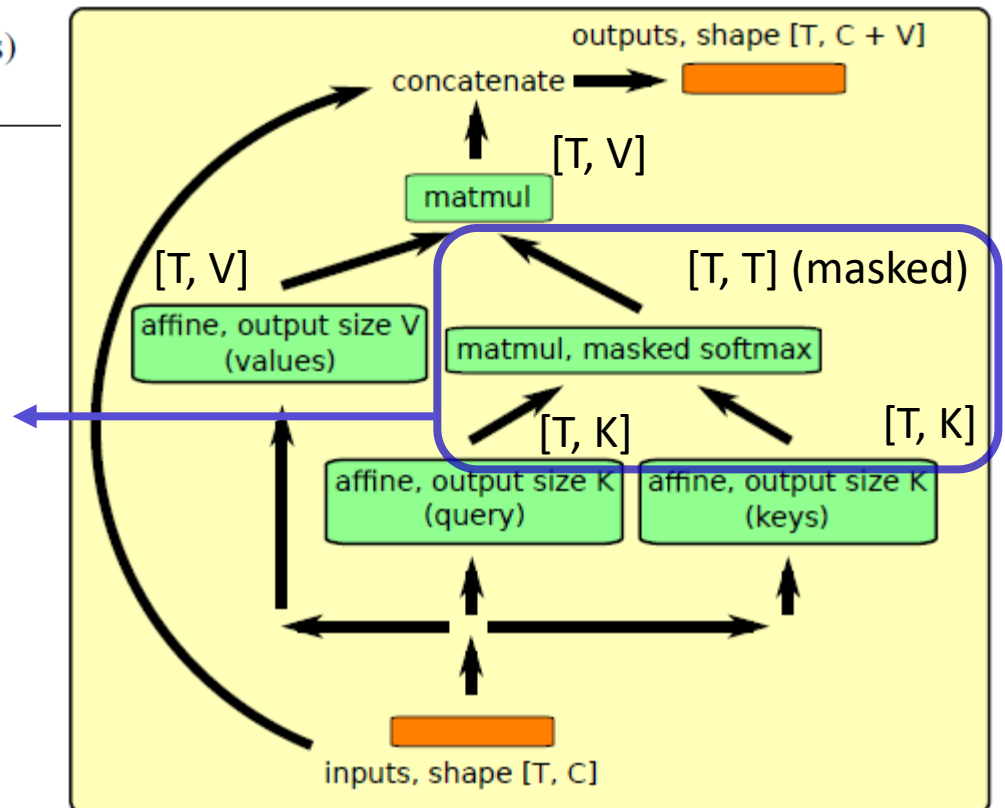


## Simple Neural Attentive meta-Learner (SNAIL)

- Two of the building blocks that compose SNAIL architectures.
- A attention block performs a causal key-value lookup and also concatenates the output to the input; they style this operation after the self-attention mechanism.

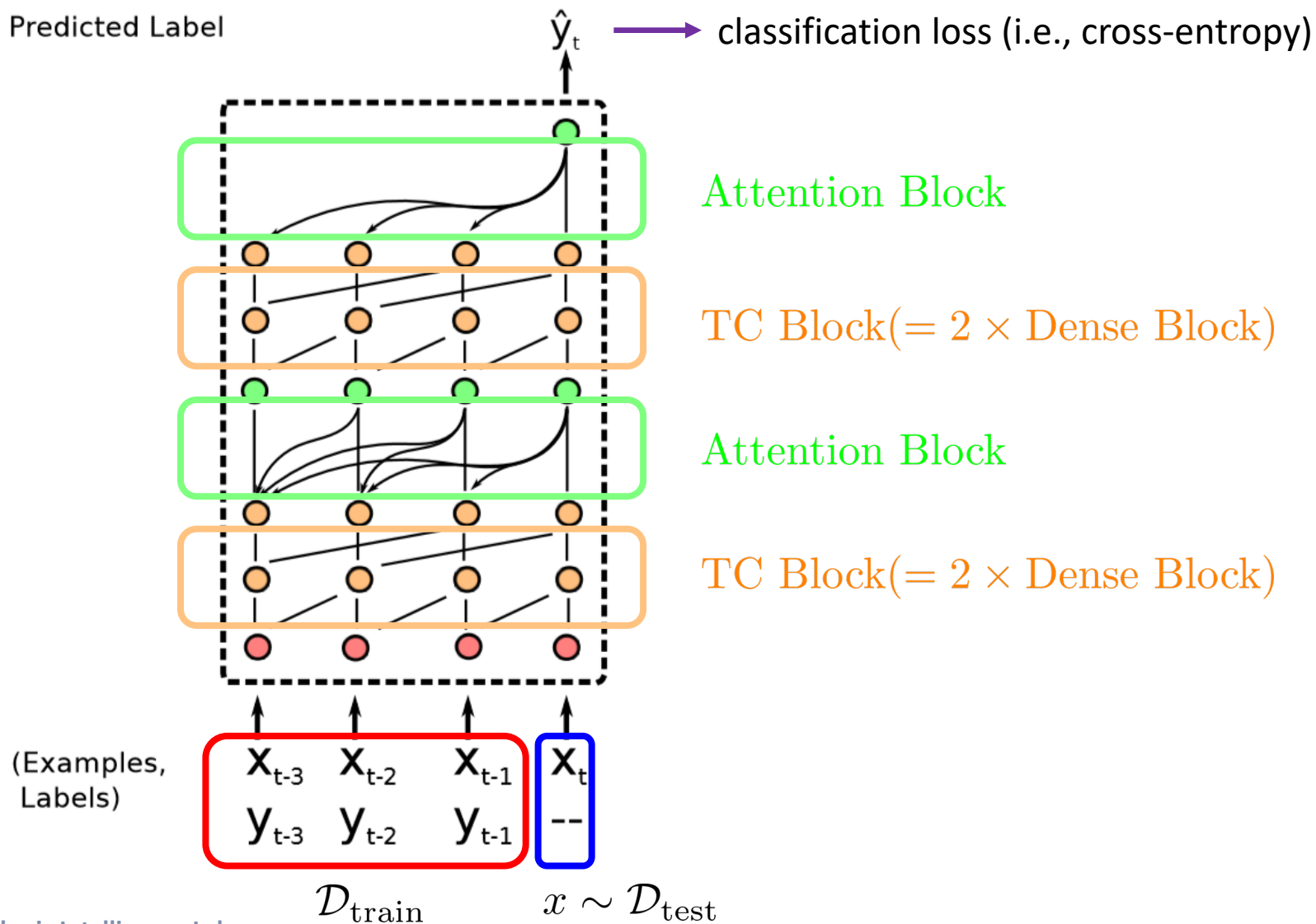
```
1: function ATTENTIONBLOCK(inputs, key size  $K$ , value size  $V$ ):  
2:   keys, query = affine(inputs,  $K$ ), affine(inputs,  $K$ )  
3:   logits = matmul(query, transpose(keys))  
4:   probs = CausallyMaskedSoftmax(logits /  $\sqrt{K}$ )  
5:   values = affine(inputs,  $V$ )  
6:   read = matmul(probs, values)  
7:   return concat(inputs, read)
```

**Self-attention** relates different positions of a single sequence in order to compute a representation



## Simple Neural Attentive meta-Learner (SNAIL)

- Overview of the SNAIL for supervised learning:



## Simple Neural Attentive meta-Learner (SNAIL)

- SNAIL outperforms state-of-the-art methods in few-shot classification tasks that are extensively hand-designed, and/or domain-specific (e.g., Matching networks [Vinyals et al. 16]).
- It significantly exceeds the performance of methods such as MANN that are similarly simple and generic.

Method	5-Way Omniglot		20-Way Omniglot	
	1-shot	5-shot	1-shot	5-shot
Santoro et al. (2016)	82.8%	94.9%	–	–
Koch (2015)	97.3%	98.4%	88.2%	97.0%
Vinyals et al. (2016)	98.1%	98.9%	93.8%	98.5%
Finn et al. (2017)	<b>98.7% ± 0.4%</b>	<b>99.9% ± 0.3%</b>	95.8% ± 0.3%	98.9% ± 0.2%
Snell et al. (2017)	97.4%	99.3%	96.0%	98.9%
Munkhdalai & Yu (2017)	<b>98.9%</b>	–	97.0%	–
SNAIL, Ours	<b>99.07% ± 0.16%</b>	<b>99.78% ± 0.09%</b>	<b>97.64% ± 0.30%</b>	<b>99.36% ± 0.18%</b>

Method	5-Way Mini-ImageNet	
	1-shot	5-shot
Vinyals et al. (2016)	43.6%	55.3%
Finn et al. (2017)	48.7% ± 1.84%	63.1% ± 0.92%
Ravi & Larochelle (2017)	43.4% ± 0.77%	60.2% ± 0.71%
Snell et al. (2017)	46.61% ± 0.78%	65.77% ± 0.70%
Munkhdalai & Yu (2017)	49.21% ± 0.96%	–
SNAIL, Ours	<b>55.71% ± 0.99%</b>	<b>68.88% ± 0.92%</b>

## 1. Introduction

- Problem setup
- Overview of meta-learning approaches

## 2. Model-based methods

- Meta-learning with External Memory
- Simple Neural Attentive meta-Learner (SNAIL)

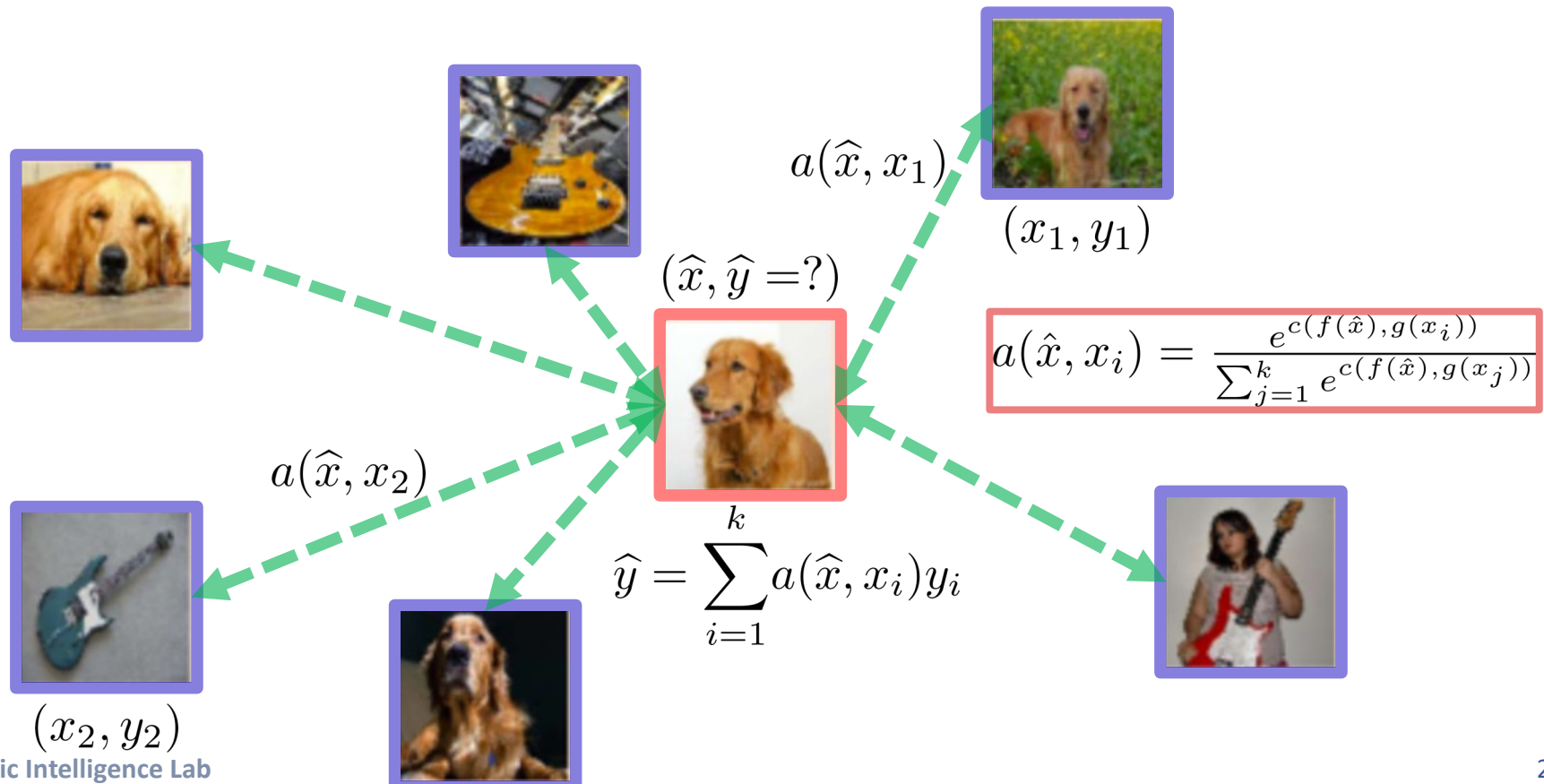
## 3. Metric-based methods

- Matching Networks
- Prototypical Networks
- Relation Networks

## 4. Optimization-based methods

- Training Meta-learner Optimizers
- Model Agnostic Meta-learning (MAML)

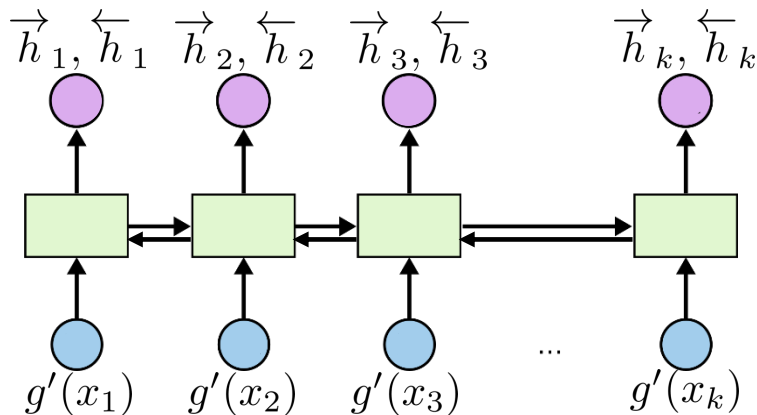
- **Matching Networks** [Vinyals et al. 16] propose to learn a shared embedding space over multiple subclassification problems.
- In this model, the neural *attention mechanism* (denoted as  $a$ ) is used as a metric function on deep features. The simplest form of  $a$  is to use the softmax over the cosine distance  $c$  with embedding functions  $f$  and  $g$ .





- Two issues of embedding functions:
  - Each element  $x_i$  gets embedded by  $g(x_i)$  *independently* of other elements in the support set  $S$ .
  - $S$  *does not modify* how we embed the test image  $\hat{x}$  through  $f$ .
- To handle this issues, authors proposed full context embeddings:  $f$  and  $g$  become  $f(\hat{x}, S)$ ,  $g(x, S)$  respectively.
- The encoding function for the elements in the support set  $S$ ,  $g(x_i, S)$ , is a bidirectional LSTM:

$$g(x_i, S) = \vec{h}_i + \overleftarrow{h}_i + g'(x_i)$$

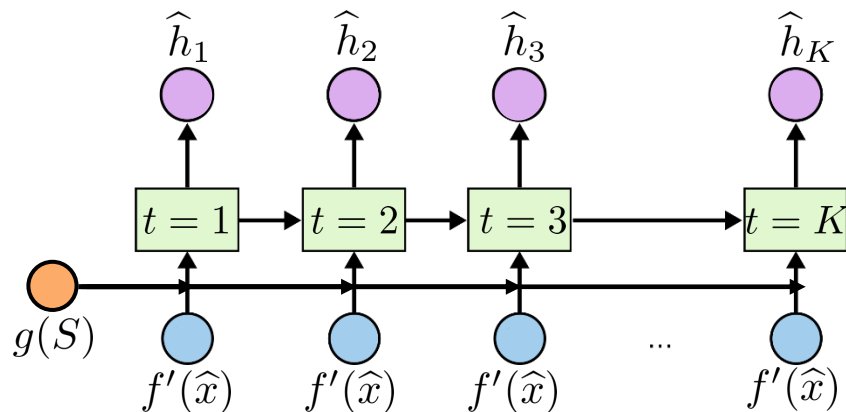


$$\begin{aligned} \vec{h}_i, \vec{c}_i &= \text{LSTM}(g'(x_i), \vec{h}_{i-1}, \vec{c}_{i-1}) \\ \overleftarrow{h}_i, \overleftarrow{c}_i &= \text{LSTM}(g'(x_i), \overleftarrow{h}_{i+1}, \overleftarrow{c}_{i+1}) \end{aligned}$$

where  $g'(x_i)$  be a convolutional neural network (e.g. VGG or Inception)

- Two issues of embedding functions:
  - Each element  $x_i$  gets embedded by  $g(x_i)$  *independently* of other elements in the support set  $S$ .
  - $S$  *does not modify* how we embed the test image  $\hat{x}$  through  $f$ .
- To handle this issues, authors proposed full context embeddings:  $f$  and  $g$  become  $f(\hat{x}, S)$ ,  $g(x, S)$  respectively.
- The encoding function for the test sample,  $f(\hat{x}, S)$ , is a LSTM with read-attention over the whole set  $S$ :

$$f(\hat{x}, S) = \text{attLSTM}(f'(\hat{x}), g(S), K) = h_K$$



$$\hat{h}_k, c_k = \text{LSTM}(f'(\hat{x}), [h_{k-1}, r_{k-1}], c_{k-1})$$

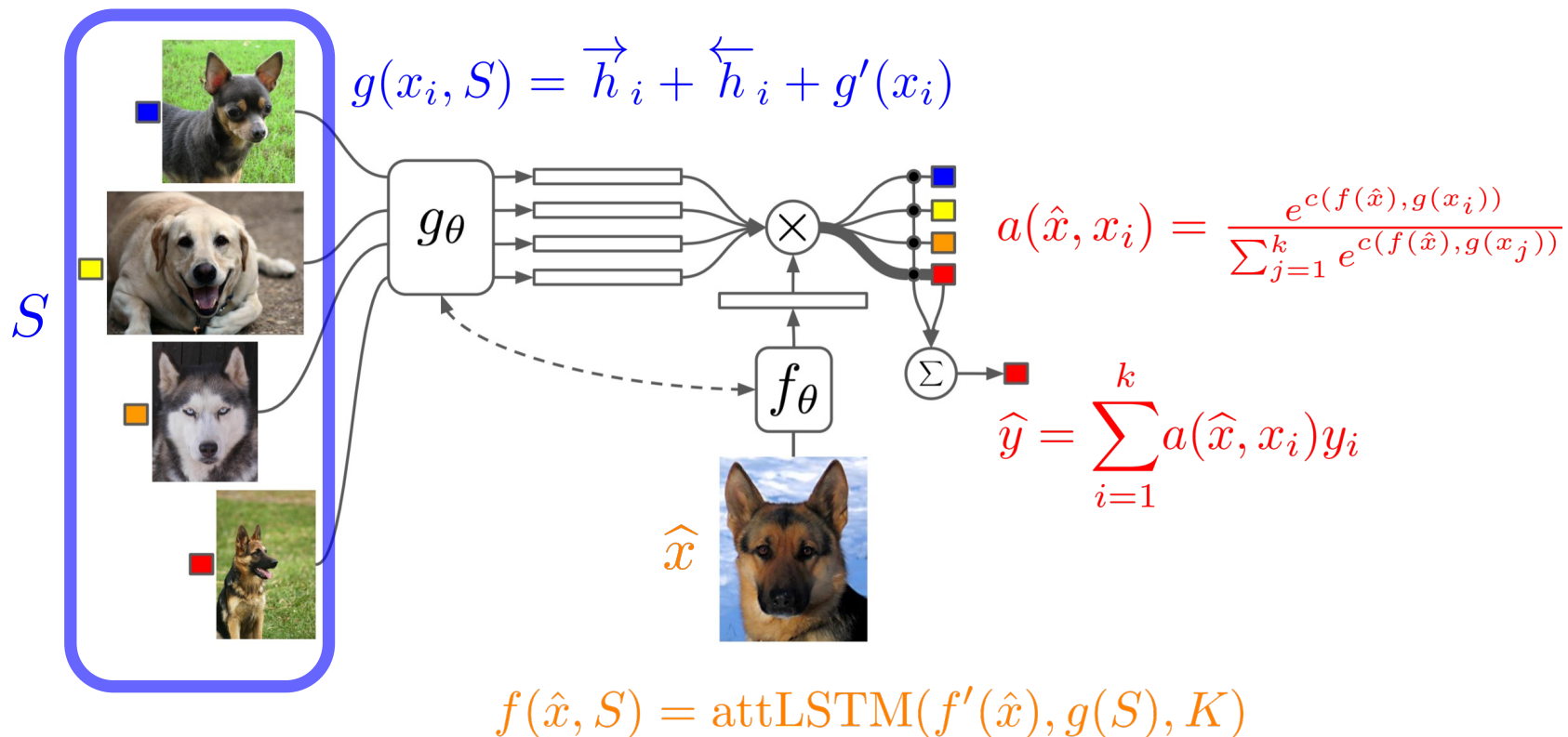
$$h_k = \hat{h}_k + f'(\hat{x})$$

$$r_{k-1} = \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i)) g(x_i)$$

$$a(h_{k-1}, g(x_i)) = \text{softmax}(h_{k-1}^T g(x_i))$$

where  $f'(\hat{x})$  be a convolutional neural network (e.g. VGG or Inception)

- The overall architecture is as follows:



- The networks are trained by maximizing log-likelihood.

$$\theta = \operatorname{argmax}_{\theta} \mathbb{E}_{L \sim \mathcal{T}} \left[ \mathbb{E}_{\mathcal{D}_{\text{train}} \sim L, \mathcal{D}_{\text{test}} \sim L} \left[ \sum_{(x, y) \in \mathcal{D}_{\text{test}}} \log P_{\theta}(y|x, S) \right] \right]$$

## Matching Networks

- Matching Networks generalize well and thus outperforms baseline classifiers and meta-learning models (MANN) on few-shot classification tasks.

Model	Matching Fn	Fine Tune	5-way Acc		20-way Acc	
			1-shot	5-shot	1-shot	5-shot
PIXELS	Cosine	N	41.7%	63.2%	26.7%	42.6%
BASLINE CLASSIFIER	Cosine	N	80.0%	95.0%	69.5%	89.1%
BASLINE CLASSIFIER	Cosine	Y	82.3%	98.4%	70.6%	92.0%
BASLINE CLASSIFIER	Softmax	Y	86.0%	97.6%	72.9%	92.3%
MANN (No CONV) [21]	Cosine	N	82.8%	94.9%	–	–
CONVOLUTIONAL SIAMESE NET [11]	Cosine	N	96.7%	98.4%	88.0%	96.5%
CONVOLUTIONAL SIAMESE NET [11]	Cosine	Y	97.3%	98.4%	88.1%	97.0%
MATCHING NETS (OURS)	Cosine	N	<b>98.1%</b>	<b>98.9%</b>	<b>93.8%</b>	<b>98.5%</b>
MATCHING NETS (OURS)	Cosine	Y	97.9%	98.7%	93.5%	<b>98.7%</b>

Table 1: Results on the Omniglot dataset.

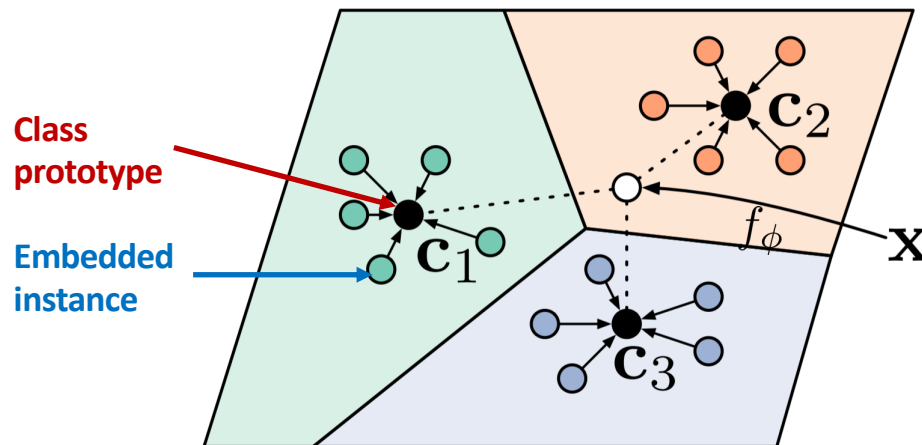
- It also works well on harder dataset, such as *miniImageNet*.

Model	Matching Fn	Fine Tune	5-way Acc	
			1-shot	5-shot
PIXELS	Cosine	N	23.0%	26.6%
BASLINE CLASSIFIER	Cosine	N	36.6%	46.0%
BASLINE CLASSIFIER	Cosine	Y	36.2%	52.2%
BASLINE CLASSIFIER	Softmax	Y	38.4%	51.2%
MATCHING NETS (OURS)	Cosine	N	<b>41.2%</b>	<b>56.2%</b>
MATCHING NETS (OURS)	Cosine	Y	<b>42.4%</b>	<b>58.0%</b>
MATCHING NETS (OURS)	Cosine (FCE)	N	<b>44.2%</b>	<b>57.0%</b>
MATCHING NETS (OURS)	Cosine (FCE)	Y	<b>46.6%</b>	<b>60.0%</b>

Table 2: Results on *miniImageNet*.

## Prototypical Networks

- **Prototypical Networks** [Snell et al. 17], is based on the idea that there exists an embedding in which points cluster around a *single prototype representation* for each class.
- They use meta-learning to learn a metric space that minimizes the distance between the prototypes and each training instance.



$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i)$$

$$p_\phi(y = k | \mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'}))}$$

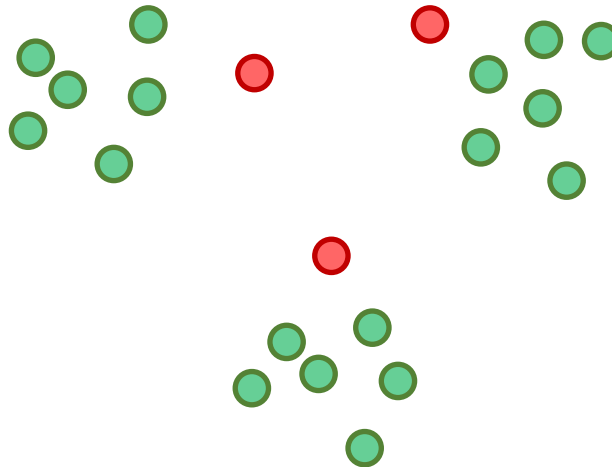
Prototype of the target class

$f_\phi$  : Embedding functions (i.e. CNN)  
 $d$  : Euclidean, or cosine distance

## Prototypical Networks

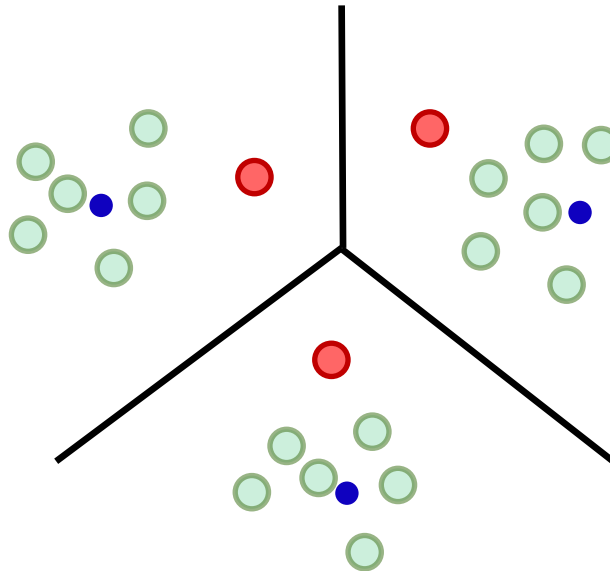
---

- Prototypical networks are trained by minimizing the negative log-probability  $J(\phi) = -\log p_\phi(y = k | \mathbf{x})$  via **episodic training**.
- The training procedure is:
  - Select a subset of classes, then choosing support examples and query examples within each class for an episode.



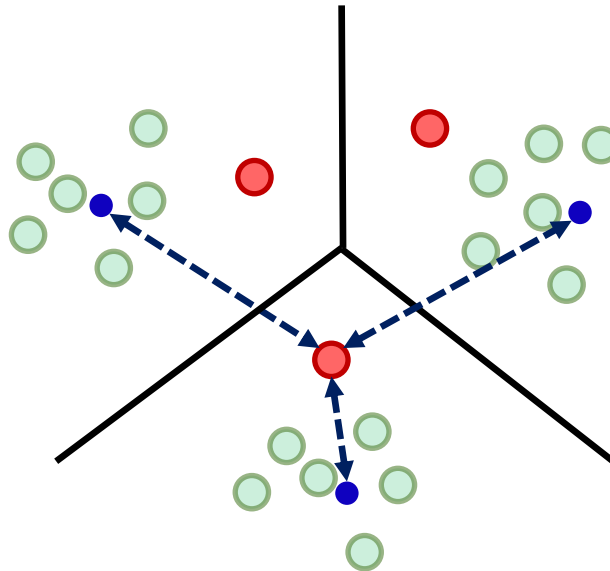
## Prototypical Networks

- Prototypical networks are trained by minimizing the negative log-probability  $J(\phi) = -\log p_{\phi}(y = k | \mathbf{x})$  via **episodic training**.
- The training procedure is:
  - Select a subset of classes, then choosing support examples and query examples within each class for an episode.
  - Compute prototype from support examples for each class.



## Prototypical Networks

- Prototypical networks are trained by minimizing the negative log-probability  $J(\phi) = -\log p_\phi(y = k | \mathbf{x})$  via **episodic training**.
- The training procedure is:
  - Select a subset of classes, then choosing support examples and query examples within each class for an episode.
  - Compute prototype from support examples for each class.
  - Compute negative likelihood loss from query examples and update networks.



$$J(\phi) = \frac{1}{|D_{\text{test}}|} \left[ \sum_{(\mathbf{x}, y) \in D_{\text{test}}} [d(f_\phi(\mathbf{x}), \mathbf{c}_y) + \log \sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'}))] \right]$$



## Prototypical Networks

- Prototypical Networks outperform Matching Networks and MAML (to be introduced) on few-shot classification tasks on Omniglot and minilImageNet dataset.
- Note that metric-based methods are stronger than others in few-shot classification tasks.

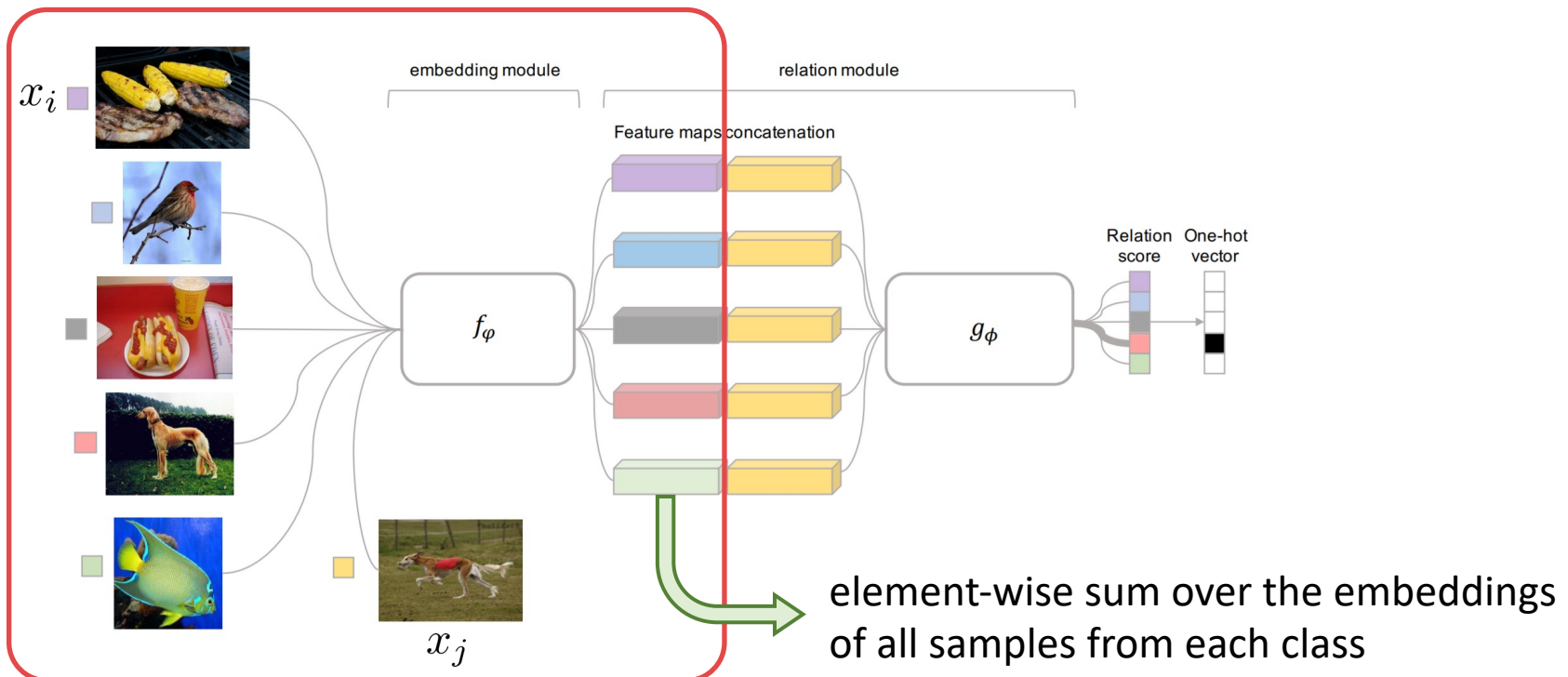
### Omniglot

Model	Dist.	Fine Tune	5-way Acc.		20-way Acc.	
			1-shot	5-shot	1-shot	5-shot
MATCHING NETWORKS [32]	Cosine	N	98.1%	98.9%	93.8%	98.5%
MATCHING NETWORKS [32]	Cosine	Y	97.9%	98.7%	93.5%	98.7%
NEURAL STATISTICIAN [7]	-	N	98.1%	99.5%	93.2%	98.1%
MAML [9]*	-	N	98.7%	<b>99.9%</b>	95.8%	<b>98.9%</b>
<b>PROTOTYPICAL NETWORKS (OURS)</b>	Euclid.	N	<b>98.8%</b>	<b>99.7%</b>	<b>96.0%</b>	<b>98.9%</b>

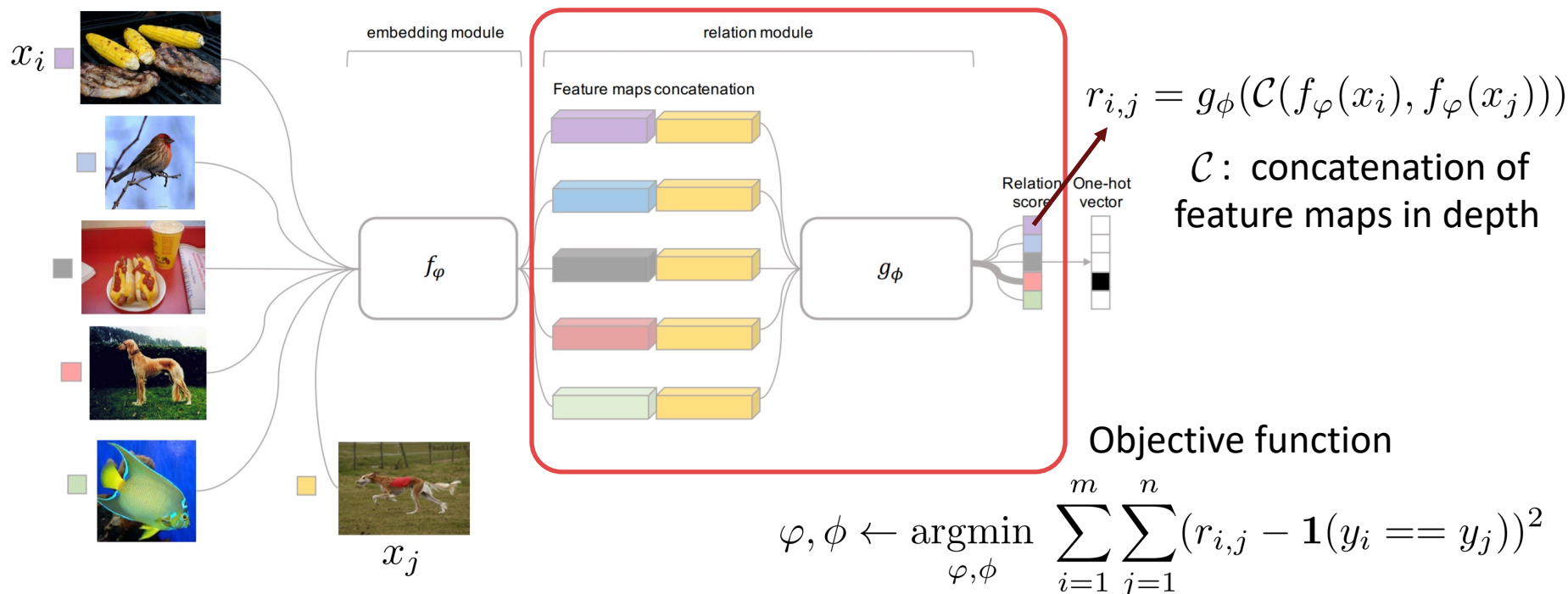
### minilImageNet

Model	Dist.	Fine Tune	5-way Acc.	
			1-shot	5-shot
BASELINE NEAREST NEIGHBORS*	Cosine	N	28.86 ± 0.54%	49.79 ± 0.79%
MATCHING NETWORKS [32]*	Cosine	N	43.40 ± 0.78%	51.09 ± 0.71%
MATCHING NETWORKS FCE [32]*	Cosine	N	43.56 ± 0.84%	55.31 ± 0.73%
META-LEARNER LSTM [24]*	-	N	43.44 ± 0.77%	60.60 ± 0.71%
MAML [9]	-	N	<b>48.70 ± 1.84%</b>	63.15 ± 0.91%
<b>PROTOTYPICAL NETWORKS (OURS)</b>	Euclid.	N	<b>49.42 ± 0.78%</b>	<b>68.20 ± 0.66%</b>

- **Relation Networks** (RN) [Sung et al. 18] is able to classify images by computing **relation scores between query images and the few examples** of each new class.
- The model consists of two modules: an embedding module  $f_\phi$  and a relation module  $g_\phi$  (both are CNNs).
  - The embedding module produces feature maps of the support set.



- **Relation Networks** (RN) [Sung et al. 18] is able to classify images by computing **relation scores between query images and the few examples** of each new class.
- The model consists of two modules: an embedding module  $f_\phi$  and a relation module  $g_\phi$  (both are CNNs).
  - The embedding module produces features maps of the support set.
  - The relation module produces a scalar in range of 0 to 1 representing **the similarity between features**, which is called relation score.



- Relation Networks outperforms Matching Networks, Prototypical Networks and MAML on few-shot learning tasks.

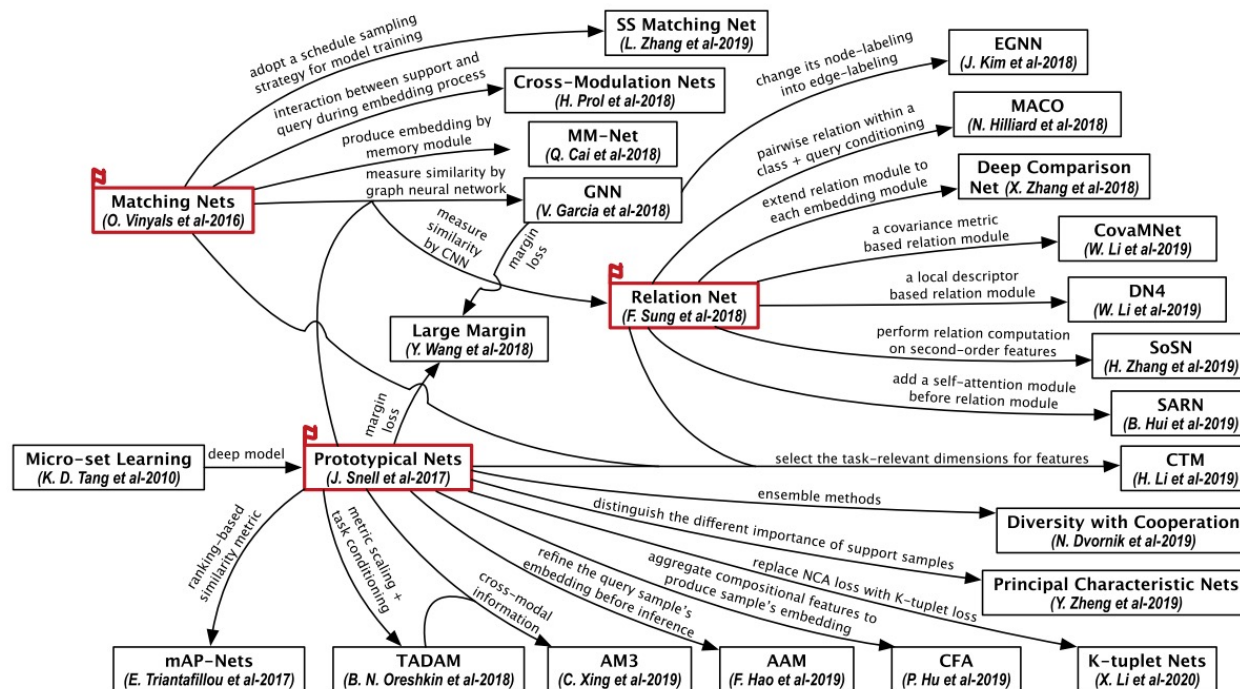
### Omniglot

Model	Fine Tune	5-way Acc.		20-way Acc.	
		1-shot	5-shot	1-shot	5-shot
MANN [32]	N	82.8%	94.9%	-	-
CONVOLUTIONAL SIAMESE NETS [20]	N	96.7%	98.4%	88.0%	96.5%
CONVOLUTIONAL SIAMESE NETS [20]	Y	97.3%	98.4%	88.1%	97.0%
MATCHING NETS [39]	N	98.1%	98.9%	93.8%	98.5%
MATCHING NETS [39]	Y	97.9%	98.7%	93.5%	98.7%
SIAMESE NETS WITH MEMORY [18]	N	98.4%	99.6%	95.0%	98.6%
NEURAL STATISTICIAN [8]	N	98.1%	99.5%	93.2%	98.1%
META NETS [27]	N	99.0%	-	97.0%	-
PROTOTYPICAL NETS [36]	N	98.8%	99.7%	96.0%	98.9%
MAML [10]	Y	98.7 ± 0.4%	99.9 ± 0.1%	95.8 ± 0.3%	98.9 ± 0.2%
RELATION NET	N	99.6 ± 0.2%	99.8 ± 0.1%	97.6 ± 0.2%	99.1 ± 0.1%

### miniImageNet

Model	FT	5-way Acc.	
		1-shot	5-shot
MATCHING NETS [39]	N	43.56 ± 0.84%	55.31 ± 0.73%
META NETS [27]	N	49.21 ± 0.96%	-
META-LEARN LSTM [29]	N	43.44 ± 0.77%	60.60 ± 0.71%
MAML [10]	Y	48.70 ± 1.84%	63.11 ± 0.92%
PROTOTYPICAL NETS [36]	N	49.42 ± 0.78%	68.20 ± 0.66%
RELATION NET	N	50.44 ± 0.82%	65.32 ± 0.70%

- Development relationship between different metric-based meta-learning methods.
- Many variants have been developed from the three representative works:
  - Matching Networks
  - Prototypical Networks
  - Relation Networks



## 1. Introduction

- Problem setup
- Overview of meta-learning approaches

## 2. Model-based methods

- Meta-learning with External Memory
- Simple Neural Attentive meta-Learner (SNAIL)

## 3. Metric-based methods

- Matching Networks
- Prototypical Networks
- Relation Networks

## 4. Optimization-based methods

- Training Meta-learner Optimizers
- Model Agnostic Meta-learning (MAML)

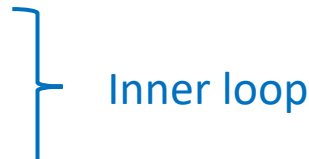
- Optimization-based meta-learning algorithms consist of **two levels of learning** (or *loops*)
  - **Inner loop**: optimizes the *base learner* (e.g., classifier)
    - **Parameters**  $\theta$  : parameters of the base learner
    - **Objective**:  $\mathcal{L}_{\text{io}}(\theta|\phi)$  (e.g., cross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

---

**Algorithm 1** Common meta-learning algorithm

---

```
1: while not done do
2:   for  $t = 1, \dots, T$  do
3:     Optimize parameters  $\theta$  of learner  $f_\theta$ 
4:      $\theta^{(t+1)} \leftarrow \theta^{(t)} - \nabla_{\theta^{(t)}} \mathcal{L}_{\text{io}}$ 
5:   end for
6:   Optimize meta-parameters  $\phi$ 
7:    $\phi \leftarrow \phi - \nabla_\phi \mathcal{L}_{\text{mo}}$ 
8: end while
```



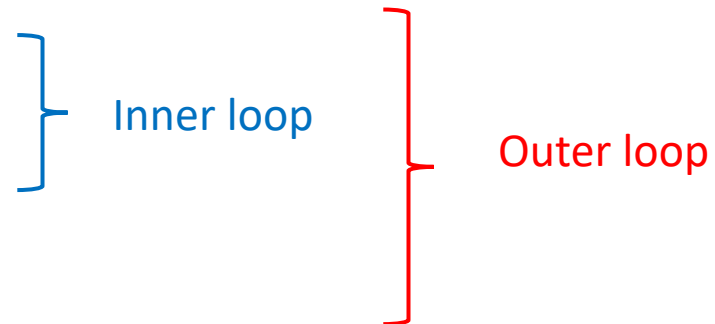
- Optimization-based meta-learning algorithms consist of **two levels of learning** (or *loops*)
  - **Inner loop**: optimizes the *base learner* (e.g., classifier)
    - **Parameters**  $\theta$  : parameters of the base learner
    - **Objective**:  $\mathcal{L}_{\text{io}}(\theta|\phi)$  (e.g., cross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

---

**Algorithm 1** Common meta-learning algorithm

---

```
1: while not done do
2:   for  $t = 1, \dots, T$  do
3:     Optimize parameters  $\theta$  of learner  $f_\theta$ 
4:      $\theta^{(t+1)} \leftarrow \theta^{(t)} - \nabla_{\theta^{(t)}} \mathcal{L}_{\text{io}}$ 
5:   end for
6:   Optimize meta-parameters  $\phi$ 
7:    $\phi \leftarrow \phi - \nabla_\phi \mathcal{L}_{\text{mo}}$ 
8: end while
```





- 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)
- $\theta$  be parameters of a neural network

- How to find the optimal  $\theta^*$  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, 83], Adagrad [Duchi et al., 11], RMSProp [Tieleman and Hinton, 12], ADAM [Kingma and Ba, 15]

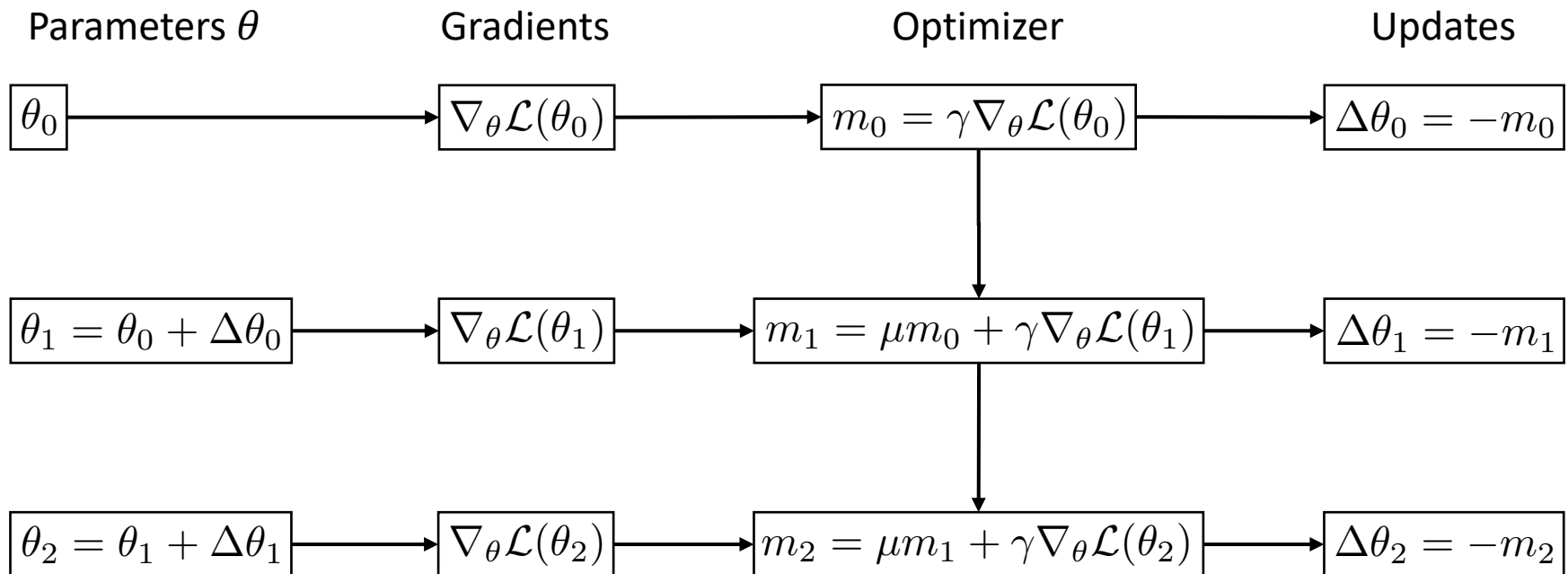
## An Example of Optimizers: SGD with Momentum

- Update rules of SGD with momentum:

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

where  $\gamma$  is a learning rate and  $\mu$  is a momentum

- Unroll the update steps



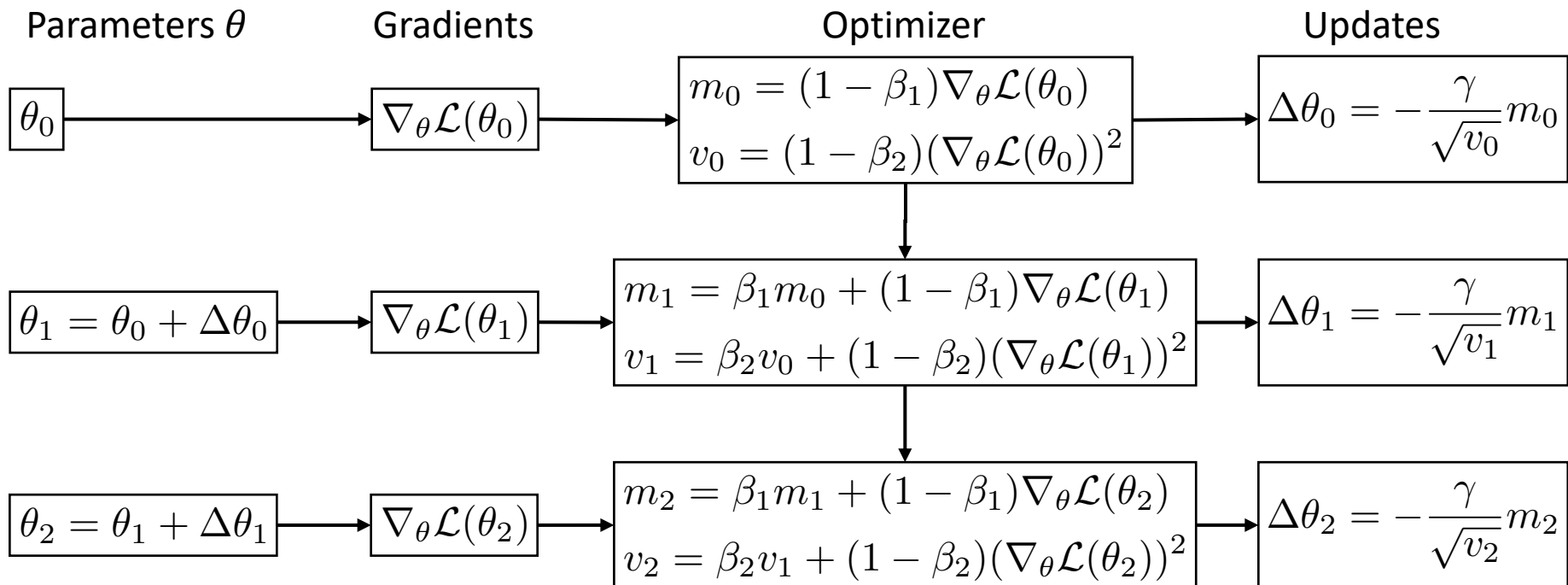
## An Example of Optimizers: ADAM

- Update rules of ADAM [Kingma and Ba, 15]:

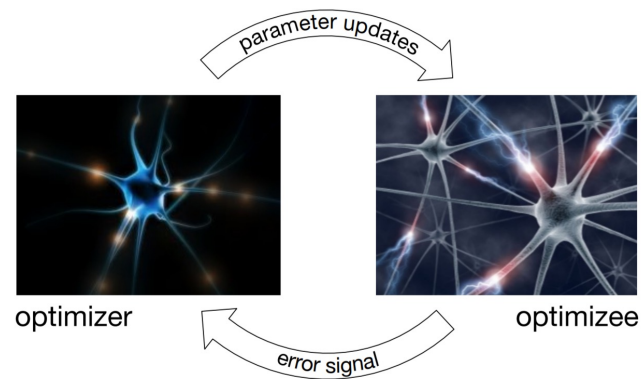
$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{v_t}} m_t \quad \begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} \mathcal{L}(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} \mathcal{L}(\theta_t))^2 \end{aligned}$$

where  $\gamma$  is a learning rate and  $\beta_1, \beta_2$  are decay rates for the moments

- Unroll the update steps



- **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., 16]



- Explicitly **model optimizers using recurrent neural networks (RNNs)**

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

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

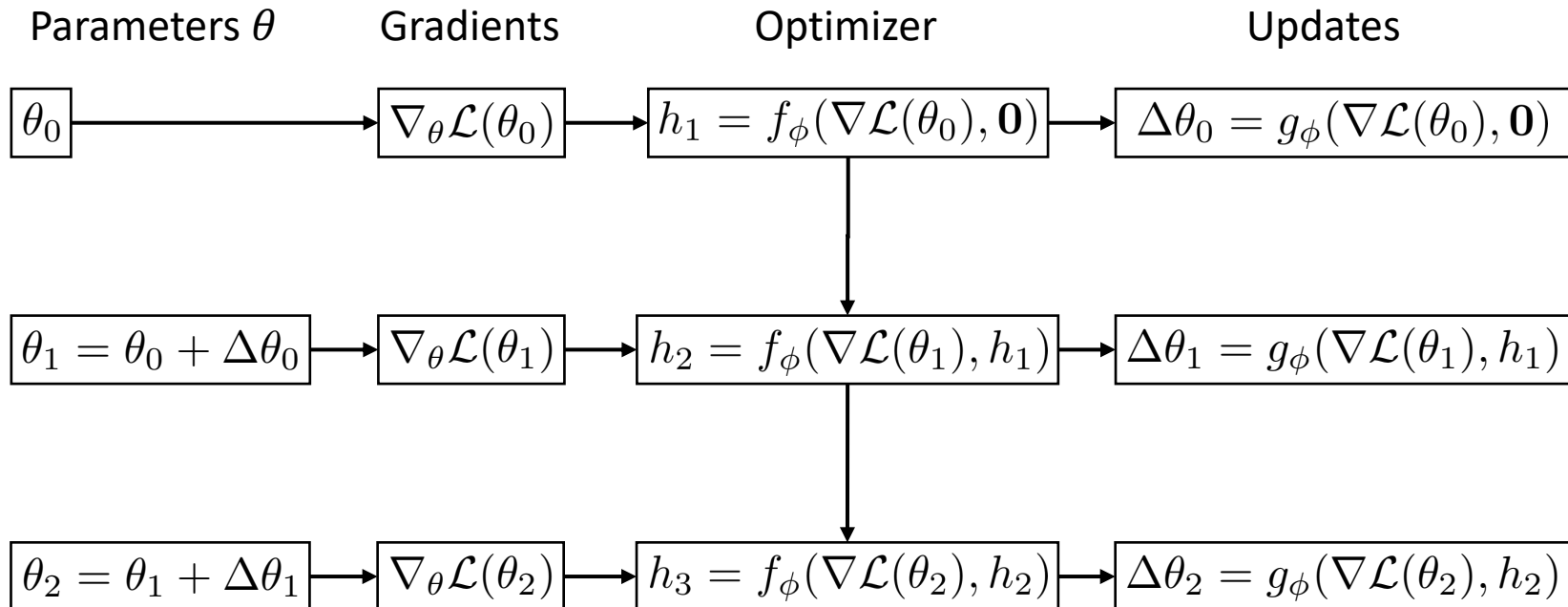
$$\phi^* = \arg \min_{\phi} \mathcal{L}(\theta_T(\phi))$$

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

- Update rules based on a RNN  $f_\phi, g_\phi$  parameterized by  $\phi$

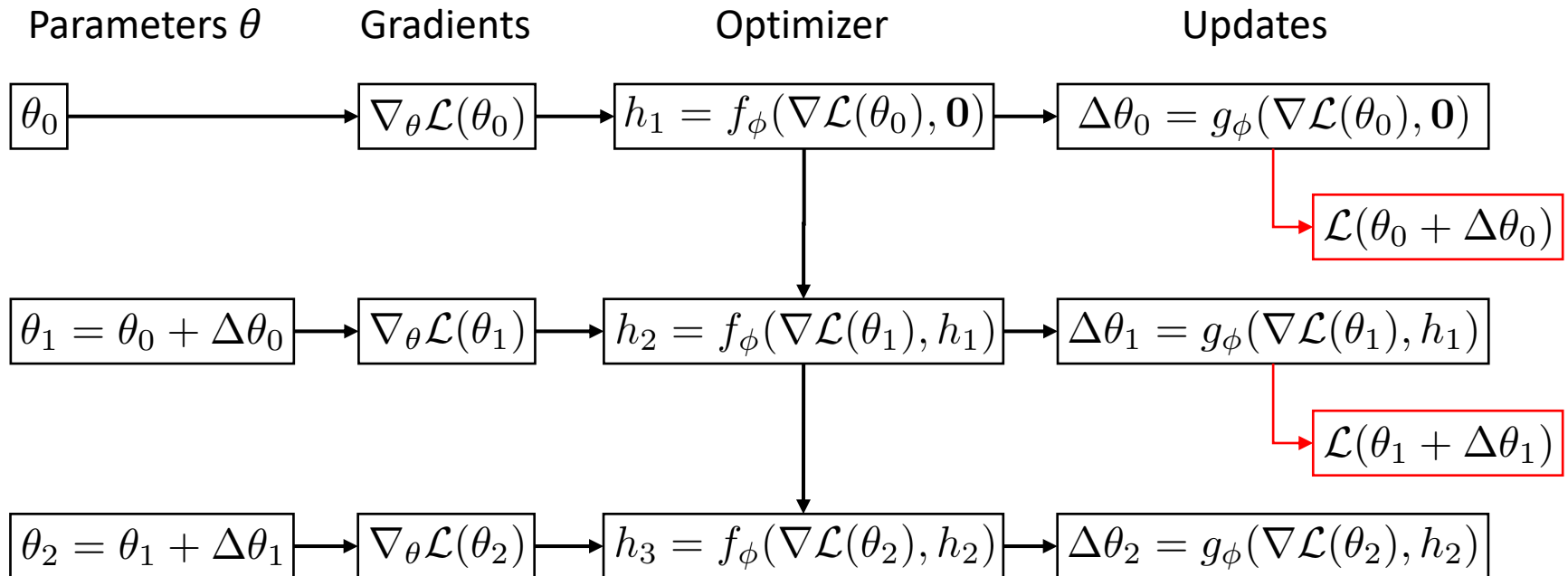
$$\theta_{t+1} = \theta_t + g_\phi(\nabla \mathcal{L}(\theta_t), h_t) \quad h_t = f_\phi(\nabla \mathcal{L}(\theta_{t-1}), h_{t-1})$$

- Inner-loop:** update the parameters  $\theta$  via the optimizer for  $T$  times



- **Objective for the RNN optimizer  $\phi$**  on the entire training trajectory

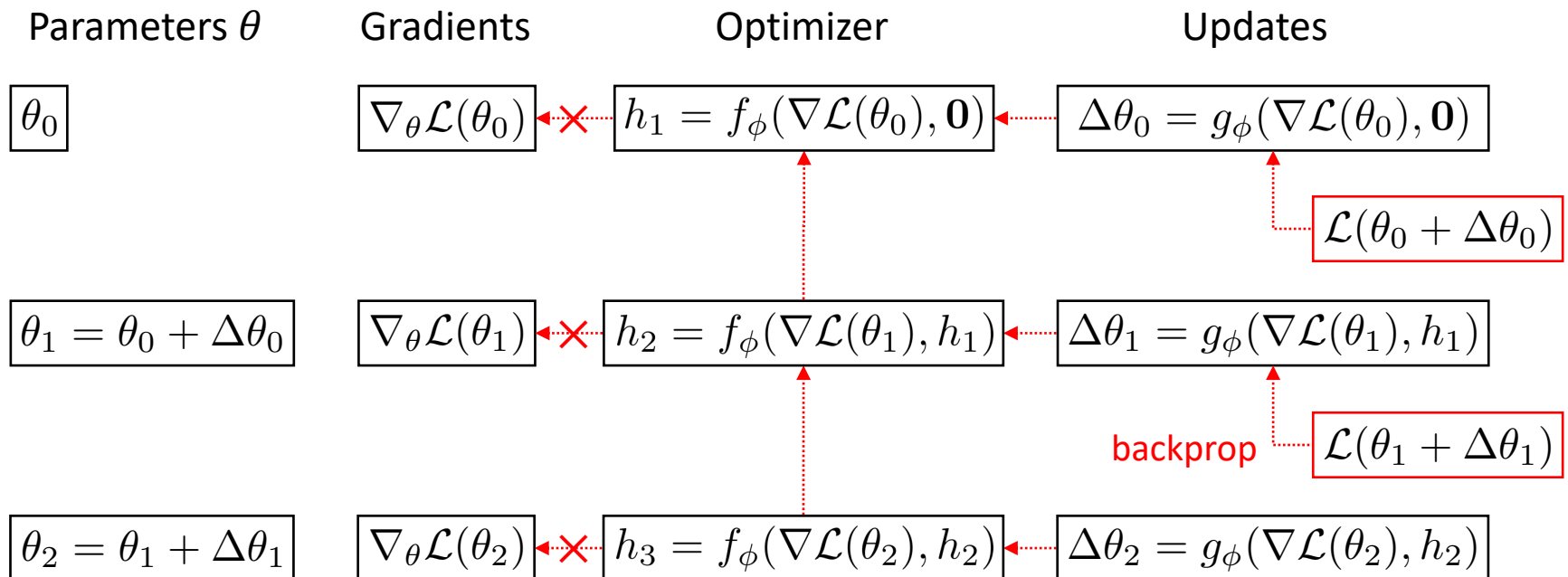
$$\mathcal{L}_{\text{meta}}(\phi) = \sum_{t=1}^T w_t \mathcal{L}(\theta_t) \quad \text{where } w_t \text{ weights for each time-step}$$



- **Objective for the RNN optimizer  $\phi$**  on the entire training trajectory

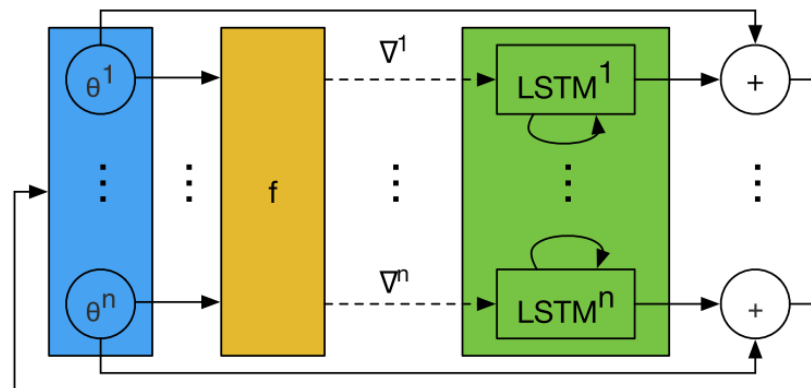
$$\mathcal{L}_{\text{meta}}(\phi) = \sum_{t=1}^T w_t \mathcal{L}(\theta_t) \quad \text{where } w_t \text{ weights for each time-step}$$

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



## Architecture of RNN Optimizer

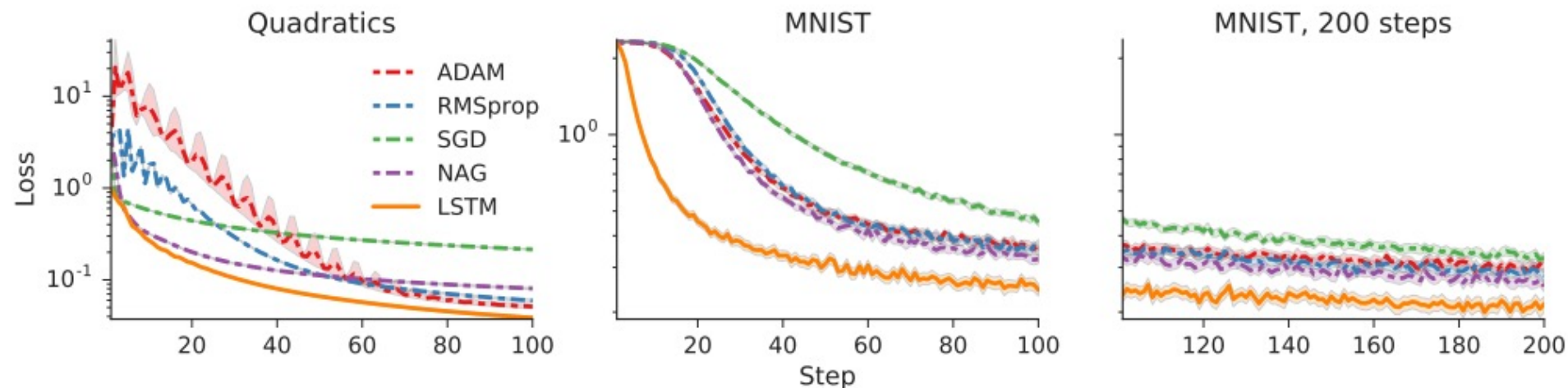
- A challenge is optimizing (at least) tens of thousands of parameters
  - Computationally not feasible with fully connected RNN architecture
- Use LSTM optimizer which **operates coordinate-wise on the parameters**
- By considering coordinate-wise optimizer
  - Able to use **small network** for optimizer
  - **Share optimizer parameters** across different parameters of the model
    - Input: gradient for single coordinate and the hidden state
    - Output: update for corresponding model parameter



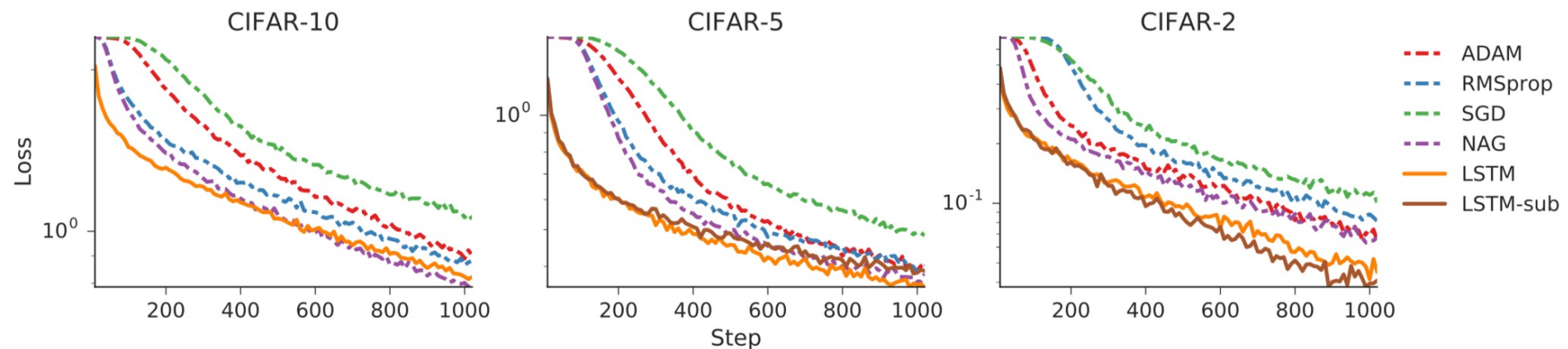


## Effectiveness of a Learned Optimizer

- 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)



- **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



- [Andrychowicz et al., 16] **compute gradients using LSTM**.

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

- [Ravi and Larochelle, 17] **formulate the whole update sequences as LSTM**:

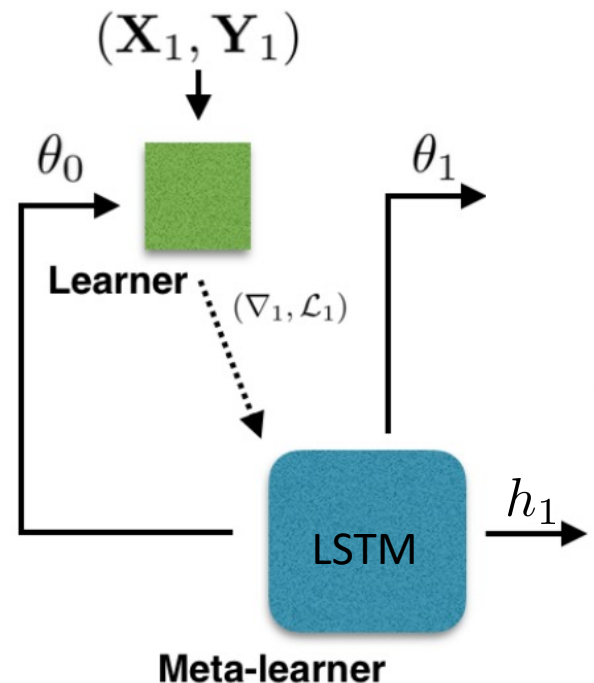
$$\begin{aligned} \theta_{t+1} &= g_\phi(\underbrace{\nabla \mathcal{L}(\theta_t), \mathcal{L}(\theta_t), \theta_t}_{\text{Inputs}}, \underbrace{h_t}_{\text{Hidden states}}) \\ &= f_{t+1} \odot \theta_t + i_{t+1} \odot \nabla \mathcal{L}(\theta_t) \end{aligned}$$

Input gate

$$i_{t+1} = \sigma(\mathbf{W}_I \cdot [\nabla \mathcal{L}(\theta_t), \mathcal{L}(\theta_t), \theta_t, i_t] + \mathbf{b}_I)$$

Forget gate

$$f_{t+1} = \sigma(\mathbf{W}_F \cdot [\nabla \mathcal{L}(\theta_t), \mathcal{L}(\theta_t), \theta_t, f_t] + \mathbf{b}_F)$$

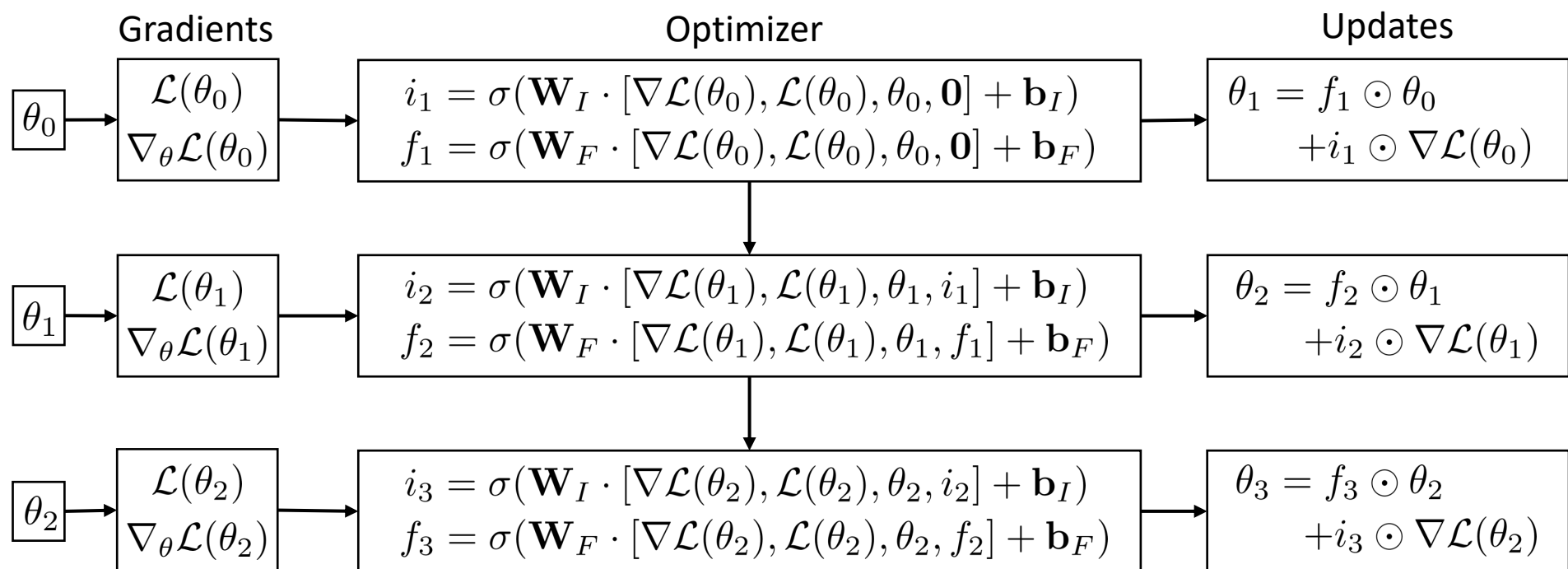


- Recall **the objective for the RNN optimizer  $\phi$**  on the entire training trajectory

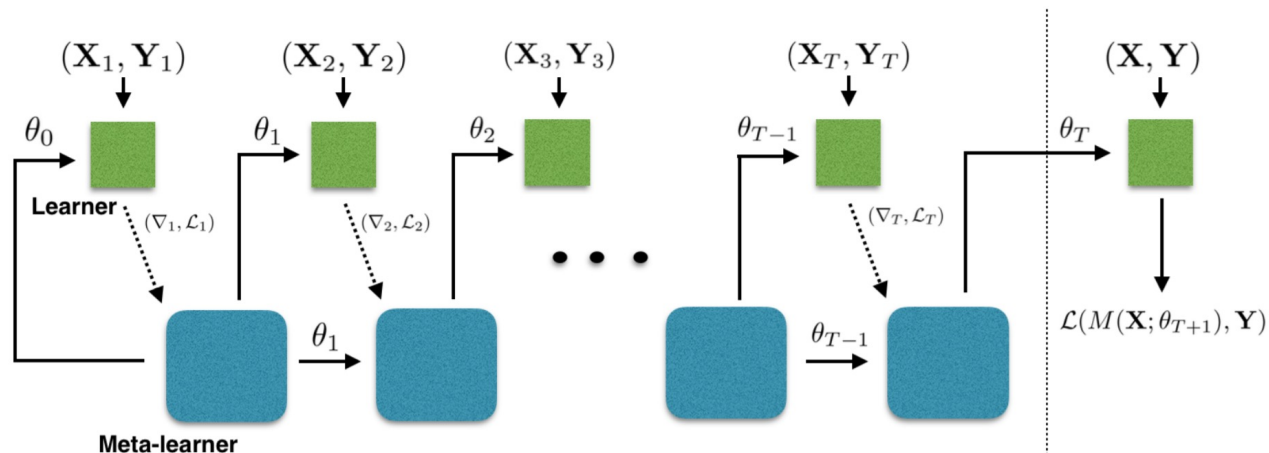
$$\mathcal{L}_{\text{meta}}(\phi) = \sum_{t=1}^T w_t \mathcal{L}(\theta_t) \quad \text{where } w_t \text{ weights for each time-step}$$

- [Ravi and Larochelle, 17] only use the last loss. (*i.e.*,  $w_t = \mathbf{1}[t = T]$ )

$$\mathcal{L}_{\text{meta}}(\phi) = \mathcal{L}(\theta_T)$$



- They used the learnable optimizer for few-shot learning.
- The meta-learning with learnable optimizer can be done by training it over multiple tasks.



Model	5-class	
	1-shot	5-shot
Baseline-finetune	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
Baseline-nearest-neighbor	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
Matching Network	<b><math>43.40 \pm 0.78\%</math></b>	$51.09 \pm 0.71\%$
Matching Network FCE	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
Meta-Learner LSTM (OURS)	<b><math>43.44 \pm 0.77\%</math></b>	<b><math>60.60 \pm 0.71\%</math></b>

- The meta-learning optimizer (Meta-learner LSTM) outperforms Matching Networks for 5-shot cases.

- **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

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

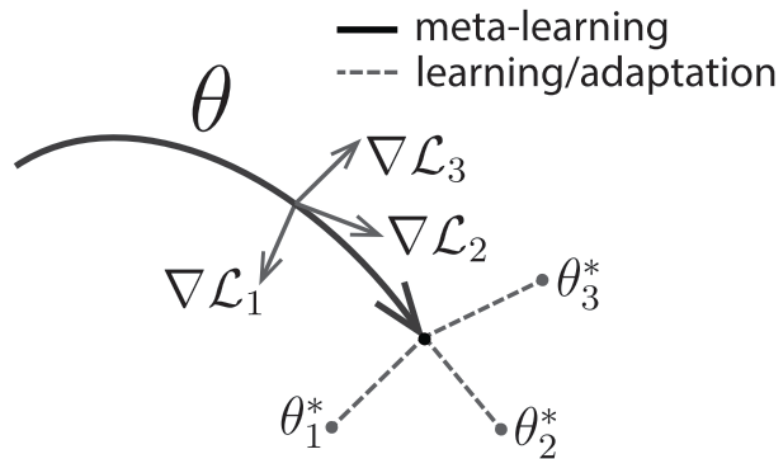
Diagram illustrating the fine-tuning process:

- A blue arrow points from the text "pre-trained parameters" to the  $\theta$  term in the equation.
- A green arrow points from the text "(new) test task" to the  $\mathcal{L}(\theta)$  term in the equation.

- **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)

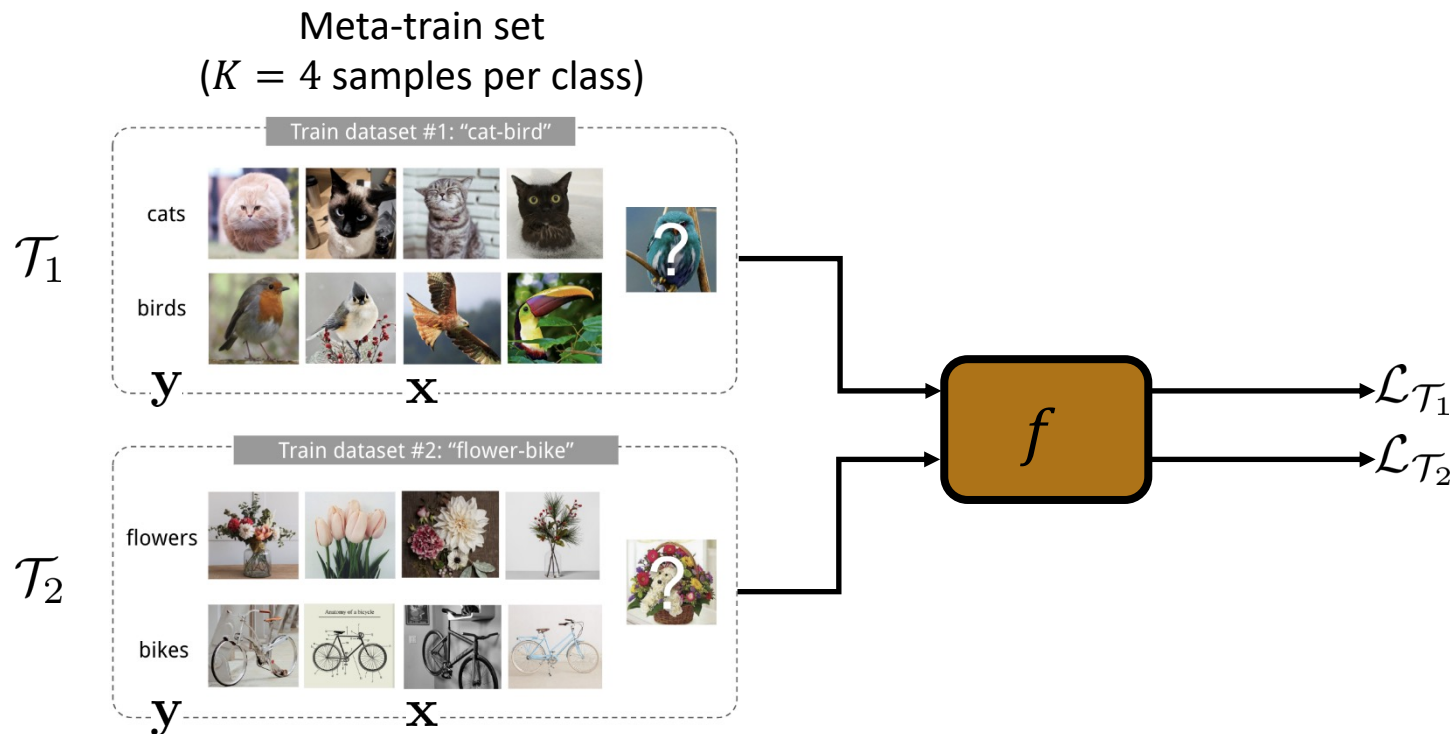
- Key idea
  - Train over **many tasks**, to learn parameter  $\theta$  that transfers well
  - Use objective that **encourage**  $\theta$  to **fast adapt** when fine-tuned with small data
  - Assumption: some representations are more transferrable than others
- Model find parameter  $\theta$  that would reduce the validation loss on each task
  - To do that, **find** (one or more steps of) **fine-tuned parameter** from  $\theta$  for each task
  - And **reduce the validation loss** at fine-tuned parameter for each task
  - Meta-update the  $\theta$  to direction **that would adapt faster** on each new task



# Model-Agnostic Meta-learning (MAML)

- 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  $\mathcal{T}_i$





- Consider a model  $f_\theta$  parameterized with  $\theta$

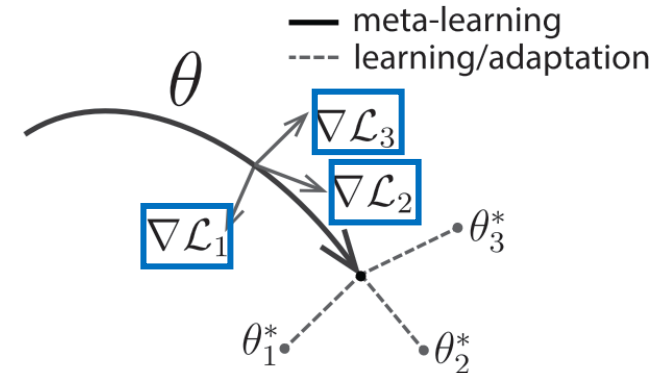
- Inner-loop

- Adapting model to a new task  $\mathcal{T}_i$

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

Where  $\alpha$  is learning rate,

- We can compute  $\theta'_i$  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  $\theta$  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:

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

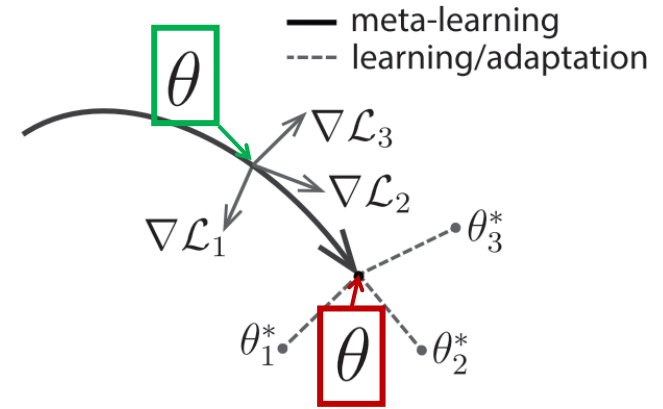
Where  $\beta$  is meta-learning rate

- Consider a model  $f_\theta$  parameterized with  $\theta$
- 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  $\alpha$  is learning rate,

- We can compute  $\theta'_i$  with one or more gradient d



$\theta$  that would adapt better than  $\theta$

- 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:

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

Where  $\beta$  is meta-learning rate

## Meta-Gradients of MAML

- MAML computes 2<sup>nd</sup> gradients
  - 1-step optimization example

Task-specificly optimized parameters

Meta-learned initial model parameters

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

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

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

## First Order Approximation of MAML

- MAML computes 2<sup>nd</sup> gradients
  - 1-step optimization example

Task-specificly optimized parameters

Meta-learned initial model parameters

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

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

- High computation cost
  - Computation cost is increased with a number of inner-loop iterations  $T$
- 
- Use 1<sup>st</sup> order approximation

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

- Ignore 2<sup>nd</sup> order terms
- Empirically show similar performance

- 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**  $\mathcal{L}_{\text{mo}}$  : 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:**  $\alpha, \beta$ : 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
```

---

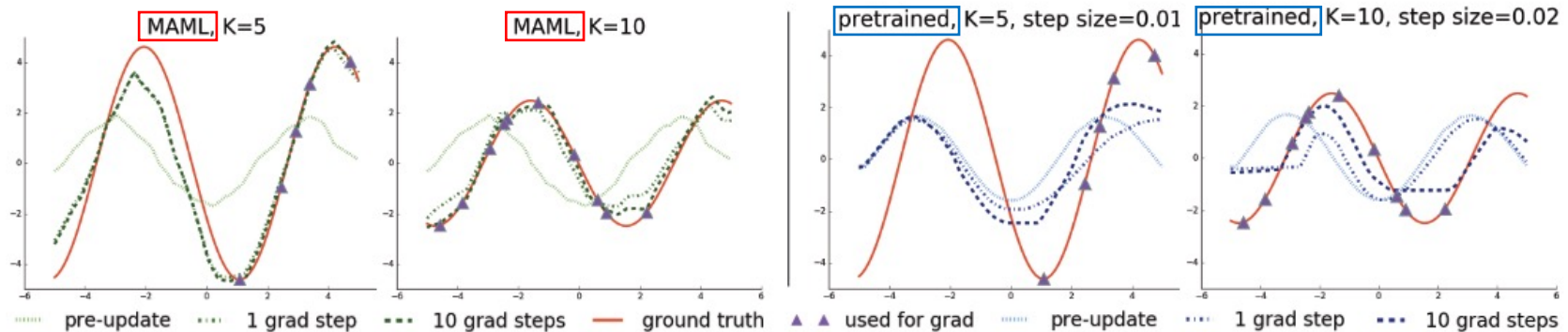
Inner loop

Outer loop

- 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

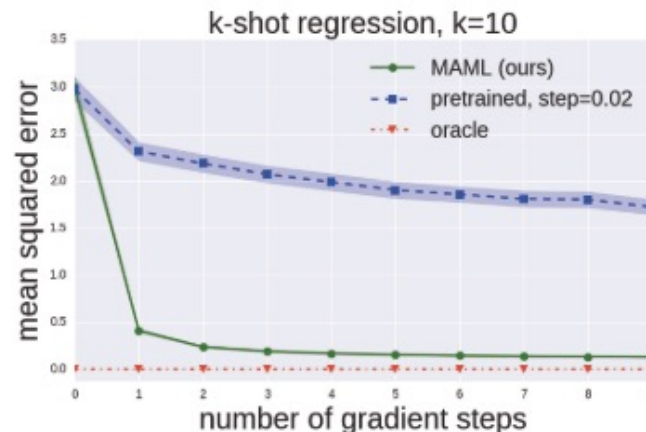
## Experiments on Few-Shot Learning Tasks

- 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)



## 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)
  - **Continue to improve** with additional gradient step
    - Not overfitted to  $\theta$  that only improves after one step
    - Learn initialization that amenable to fast adaptation





## Experiments on Few-Shot Learning Tasks

- Few-shot classification experiments
  - Omniglot

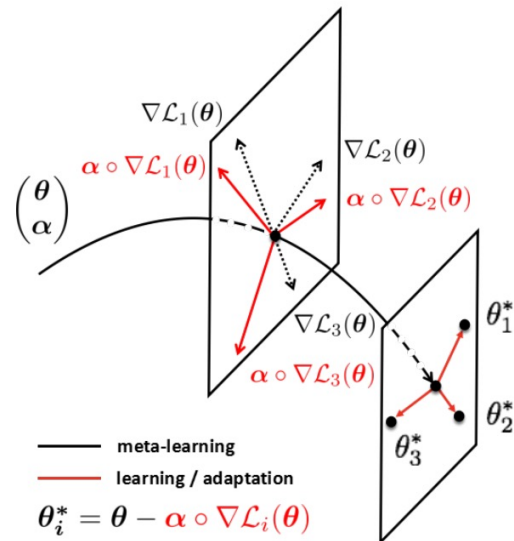
	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Omniglot (Lake et al., 2011)				
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
<b>MAML, no conv (ours)</b>	<b>89.7 ± 1.1%</b>	<b>97.5 ± 0.6%</b>	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	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%
<b>MAML (ours)</b>	<b>98.7 ± 0.4%</b>	<b>99.9 ± 0.1%</b>	<b>95.8 ± 0.3%</b>	<b>98.9 ± 0.2%</b>

- Mini-ImageNet

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

- 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., 17]
  - First-order approximation of MAML [Nichol, et. al., 18]
  - Probabilistic MAML [Finn, et. al., 18]
  - Visual imitation learning [Finn, et. al., 17]
  - LEO [Rusu, et al., 18]
  - MT-NET [Lee, et al., 18]
  - CAVIA [Zintgraf, et al., 19]

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



## Experimental Results on Few-Shot Regression

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

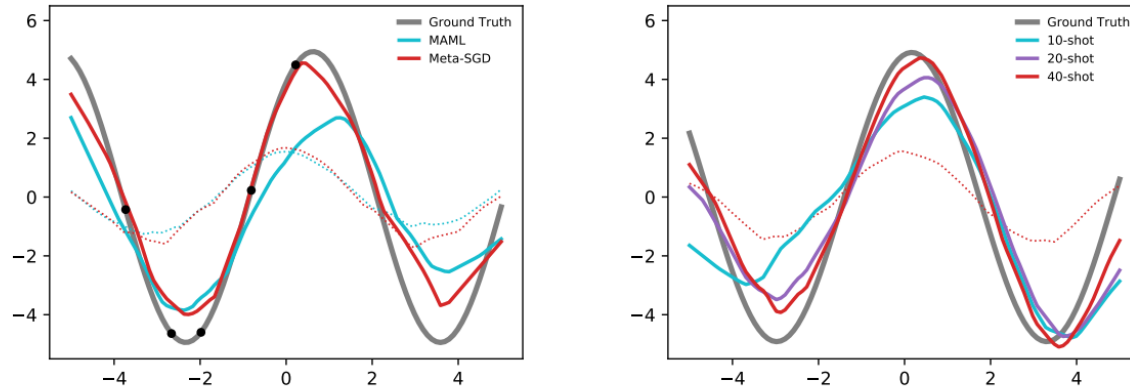


Figure 3: **Left:** Meta-SGD vs MAML on 5-shot regression. Both initialization (dotted) and result after one-step adaptation (solid) are shown. **Right:** Meta-SGD (10-shot meta-training) performs better with more training examples in meta-testing.

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	$1.13 \pm 0.18$	$0.85 \pm 0.14$	$0.71 \pm 0.12$
	Meta-SGD	<b><math>0.90 \pm 0.16</math></b>	<b><math>0.63 \pm 0.12</math></b>	<b><math>0.50 \pm 0.10</math></b>
10-shot training	MAML	$1.17 \pm 0.16$	$0.77 \pm 0.11$	$0.56 \pm 0.08$
	Meta-SGD	<b><math>0.88 \pm 0.14</math></b>	<b><math>0.53 \pm 0.09</math></b>	<b><math>0.35 \pm 0.06</math></b>
20-shot training	MAML	$1.29 \pm 0.20$	$0.76 \pm 0.12$	$0.48 \pm 0.08$
	Meta-SGD	<b><math>1.01 \pm 0.17</math></b>	<b><math>0.54 \pm 0.08</math></b>	<b><math>0.31 \pm 0.05</math></b>

## 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	<b><math>99.53 \pm 0.26\%</math></b>	<b><math>99.93 \pm 0.09\%</math></b>	<b><math>95.93 \pm 0.38\%</math></b>	<b><math>98.97 \pm 0.19\%</math></b>

- 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	<b><math>50.47 \pm 1.87\%</math></b>	<b><math>64.03 \pm 0.94\%</math></b>	<b><math>17.56 \pm 0.64\%</math></b>	<b><math>28.92 \pm 0.35\%</math></b>

- 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*

- Recently, to improve generalization in meta-learning, task augmentation approaches have been proposed. [Yao et al., 21; Yao et al., 22; Ni et al., 22]
  - Analogous to data augmentation in standard supervised learning
- One notable example is Meta learning with task interpolation (MLTI) [Yao et al., 22]
  - linearly combines features/hidden representation and labels of samples from both the support and query sets.

- Given hidden representations  $(\mathbf{H}_i^{s(q),l}, \mathbf{H}_j^{s(q),l})$ , and corresponding labels  $(\mathbf{Y}_i^{s(q)}, \mathbf{Y}_j^{s(q)})$  on the support (query) sets,

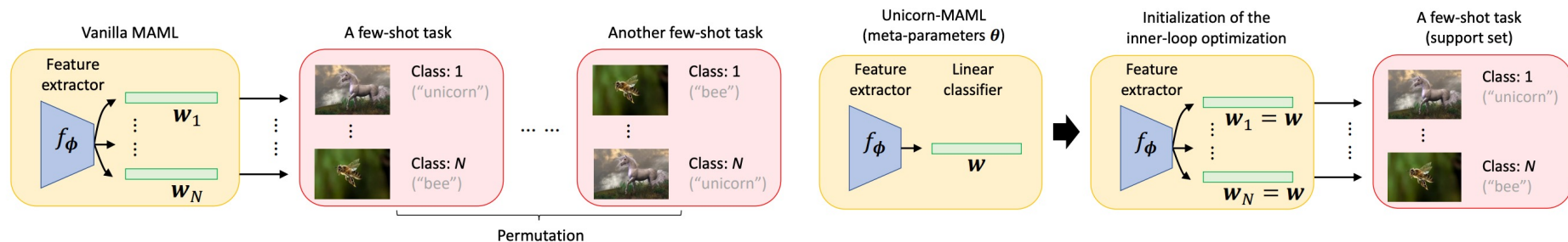
$$\tilde{\mathbf{H}}_{cr}^{s,l} = \lambda \mathbf{H}_i^{s,l} + (1 - \lambda) \mathbf{H}_j^{s,l}, \quad \tilde{\mathbf{Y}}_{cr}^{s,l} = \lambda \mathbf{Y}_i^s + (1 - \lambda) \mathbf{Y}_j^s,$$

$$\tilde{\mathbf{H}}_{cr}^{q,l} = \lambda \mathbf{H}_i^{q,l} + (1 - \lambda) \mathbf{H}_j^{q,l}, \quad \tilde{\mathbf{Y}}_{cr}^{q,l} = \lambda \mathbf{Y}_i^q + (1 - \lambda) \mathbf{Y}_j^q,$$

- where  $\lambda \in [0, 1]$  is sampled from a Beta distribution  $\text{Beta}(\alpha, \beta)$
- If label space for two given tasks are different
  - interpolate features/hidden representation as usual,
  - Re-assign a new label to the interpolated class. (i.e., one new label for each mixed sample)

## Further improvement over MAML

- [Ye et al., 22] find that MAML is sensitive to the class label assignments during meta-testing, and propose permutation-invariant version of MAML: Unicorn-MAML
  - A vanilla MAML learns the initialization of feature extractor  $\phi$  and classification head for each class  $\{\mathbf{w}_c\}_{c=1}^N$ 
    - However, a few-shot task may consist of the same set of semantic classes but in different permutations of class label assignments, leading to a larger variance in meta-testing accuracy.
  - Unicorn-MAML, besides learning  $\phi$ , learns only a single weight vector  $\mathbf{w}$ , which initialize all the  $N$  weight vectors  $\{\mathbf{w}_c\}_{c=1}^N$  to force permutation-invariance.



- Meanwhile, some approaches have been proposed recently to mitigate short-horizon bias of meta-learning [Wu et al., 18; Metz et al., 19].
  - Bootstrapped meta-learning [Flennerhag et al., 22]
  - Continuous-time meta-learning (COMLN) [Deleu et al., 22]



- Meta-learning is a study about learning the learning rules
  - Make learner which is fast adapted to unseen task with only few examples
- **Metric-based meta-learning**
  - The key idea is to learn a metric or distance function on deep neural features over objects
- **Metric-based meta-learning**
  - The key idea is to learn a metric or distance function on deep neural features over objects
- **Optimization-based meta-learning**
  - The key idea is to adjust the optimization algorithm (e.g. gradient descent) with a few examples
- It is applied for many other fields as well
  - Hyperparameter optimization
  - Neural network architecture search
  - Reinforcement learning

## References

---

- [Andrychowicz, et. al., 16] Learning to learn by gradient descent by gradient descent, NIPS 2016  
<https://arxiv.org/abs/1606.04474>
- [Vinyals, et. al., 16] Matching networks for one shot learning, NIPS 2016  
<https://arxiv.org/abs/1606.04080>
- [Santoro, et. al., 16] One-shot learning with memory-augmented neural networks, ICML 2016  
<https://arxiv.org/abs/1605.06065>
- [Koch, et. al., 15] Siamese neural networks for one-shot image recognition, ICML workshop 2015  
<https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf>
- [Ravi and Larochelle, 17] Optimization as a model for few-shot learning, ICLR 2017  
<https://openreview.net/pdf?id=rJY0-KcII>
- [Lake, et. al., 15] 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., 17] Prototypical networks for few-shot learning, NIPS 2017  
<http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning>
- [Mishra, et. al., 18] A simple neural attentive meta-learner, ICLR 2018  
<https://openreview.net/pdf?id=B1DmUzWAW>
- [Lemke, et. al., 15] 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., 09] 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., 18] Learning unsupervised learning rules, 2018

<https://arxiv.org/abs/1804.00222>

[Li and Malik, 17] Learning to optimize, ICLR 2017

<https://arxiv.org/pdf/1606.01885.pdf>

[Wichrowska, et. al., 17] Learned optimizers that scale and generalize, ICML 2017

<https://arxiv.org/pdf/1703.04813.pdf>

[Nichol, et. al., 18] On first-order meta-learning algorithms, 2018

<https://arxiv.org/abs/1803.02999>

[Finn, et. al., 17] Model-agnostic meta-learning, ICML 2017

<https://arxiv.org/abs/1703.03400>

[Finn, et. al., 18] Probabilistic model-agnostic meta-learning, NIPS 2018

<https://arxiv.org/abs/1806.02817>

[Finn, et. al., 17] One-Shot Visual Imitation Learning via Meta-learning, CoRL 2017

<https://arxiv.org/abs/1709.04905>

[Metz, et. al., 18] Learned optimizers that outperform SGD on wall-clock and test loss, 2018

<https://arxiv.org/abs/1810.10180>

[Li, et. al., 17] Meta-SGD: Learning to learn quickly for few-shot learning

<https://arxiv.org/pdf/1707.09835.pdf>

## References

---

[Nesterov, 83] A method of solving a convex programming problem with convergence rate  $o(1/k^2)$ , Soviet Mathematics Doklady, 1983

[Duchi et al., 11] Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011  
<http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>

[Tieleman and Hinton, 12] 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, 15] Adam: A method for stochastic optimization, ICLR 2015  
<https://arxiv.org/pdf/1412.6980.pdf>

[Wolpert and Macready, 97] No free lunch theorems for optimization, Transactions on Evolutionary Computation, 1997  
<https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf>

[Finn, et al., 18] Meta-learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm, ICLR 2018  
<https://arxiv.org/pdf/1710.11622.pdf>

[Sung, et. al., 18] Learning to Compare: Relation Network for Few-Shot Learning, CVPR 2018  
<https://arxiv.org/pdf/1711.06025.pdf>

[Grant, et. al., 18] Recasting Gradient-Based Meta-learning as Hierarchical Bayes, ICLR 2018  
<https://arxiv.org/pdf/1801.08930.pdf>

[Kim, et. al., 18] Auto-Meta: Automated Gradient Based Meta Learner Search, NIPS 2018  
<https://arxiv.org/pdf/1806.06927.pdf>

## References

---

[Lee, et. al., 18] Gradient-Based Meta-learning with Learned Layerwise Metric and Subspace, ICML 2018

<https://arxiv.org/pdf/1902.04552.pdf>

[A. Rusu, et. al., 19] Meta-learning with Latent Embedding Optimization, ICLR 2019

<https://arxiv.org/pdf/1807.05960.pdf>

[Allen, et. al., 19] Infinite Mixture Prototypes for Few-Shot Learning, ICML 2019

<https://arxiv.org/pdf/1902.04552.pdf>

[Zintgraf, et. al., 19] Fast Context Adaptation via Meta-Learning, ICML 2019

<https://arxiv.org/abs/1810.03642.pdf>

## References

---

- [Chen, et. al., 18] Semantic feature augmentation in few-shot learning, IEEE Trans. Image Process, 2019  
<https://arxiv.org/abs/1804.05298>
- [Wang et al., 18] Low-Shot Learning from Imaginary Data, CVPR, 2018  
<https://arxiv.org/abs/1801.05401>
- [Schwartz, et. al., 18]  $\Delta$ -encoder: an effective sample synthesis method for few-shot object recognition, NeurIPS, 2018  
<https://arxiv.org/abs/1806.04734>
- [Chen et al., 19] Image Deformation Meta-Networks for One-Shot Learning, CVPR, 2019  
<https://arxiv.org/abs/1905.11641>
- [Li, et. al., 20] Adversarial Feature Hallucination Networks for Few-Shot Learning, CVPR, 2020  
<https://arxiv.org/abs/2003.13193>
- [Lu, et. Al., 20] Learning from Very Few Samples: a Survey, Arxiv, 2020  
<https://arxiv.org/abs/2009.02653>
- [Chen et al. 19] A closer look at few-shot classification, ICLR, 2019  
<https://arxiv.org/abs/1904.04232>
- [Dhillon et al. 19] A baseline for few-shot image classification, ICLR, 2020  
<https://arxiv.org/abs/1909.02729>
- [Tian et al., 19] Rethinking Few-Shot Image Classification: A Good Embedding is All You Need?, Arxiv, 2020  
<https://arxiv.org/abs/2003.11539>
- [Furlanello et al. 18] Born Again Neural Networks, ICML, 2018  
<https://arxiv.org/abs/1805.04770>
- [Chen et al., 20] Big Self-Supervised Models are Strong Semi-Supervised Learners, NeurIPS, 2020  
<https://arxiv.org/abs/2006.10029>