Meta Learning

Recent Advances in Deep Learning (AI602)

Lecture 15

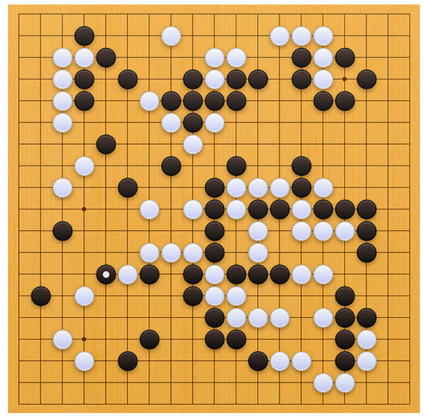
Slide made by

Jongjin Park and Sangwoo Mo
KAIST Graduate School of AI

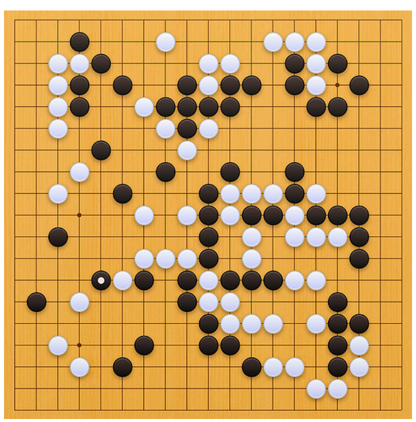
Table of Contents

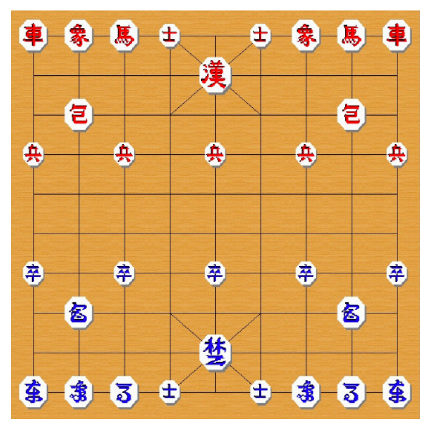
Table of Contents

- **Learning:** The model learns to solve a problem



- **Meta-learning:** The model learns to learn (fast adapt) new problems

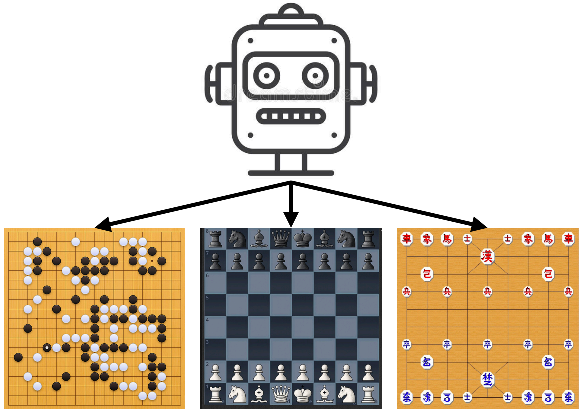




New Boardgame?

- **Multi-task learning:**
  - Given a pre-defined set of tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$ (and corresponding loss functions $\{\mathcal{L}_i\}$), learn a single model f that solves all tasks simultaneously
  - Formally, the objective is given by

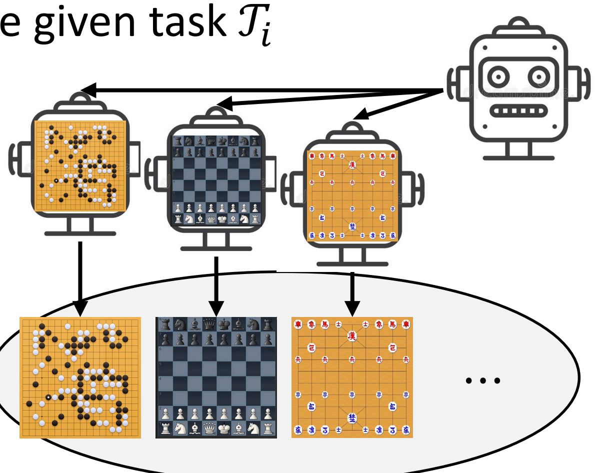
$$\underset{f}{\operatorname{argmin}} \sum_{k=1}^{K} \mathcal{L}_k(\mathcal{T}_k; f)$$



- **Meta-learning:**
  - For each task $\mathcal{T}_i$ from a task distribution $p(\mathcal{T})$, learn a **meta-model** f that (quickly) learns a task-specific model $f_i \coloneqq f(\cdot\,|\mathcal{T}_i)$ that solves the given task $\mathcal{T}_i$
  - Formally, the objective is given by

$$\underset{f}{\operatorname{argmin}}\ \mathbb{E}_{\mathcal{T}_i}\ \mathcal{L}_i(\mathcal{T}_i; f_i)$$

Key difference: **adaptation**



- **Multi-task learning:**
  - Given a pre-defined set of tasks $\{\mathcal{T}_1, \dots, \mathcal{T}_K\}$ (and corresponding loss functions $\{\mathcal{L}_i\}$), learn a single model f that solves all tasks simultaneously
  - Formally, the objective is given by

$$\operatorname*{argmin}_f \sum_{k=1}^{K} \mathcal{L}_k(\mathcal{T}_k; f)$$

- **Meta-learning:**
  - For each task $\mathcal{T}_i$ from a task distribution $p(\mathcal{T})$, learn a **meta-model** f that (quickly) learns a task-specific model $f_i \coloneqq f(\cdot\,|\mathcal{T}_i)$ that solves the given task $\mathcal{T}_i$
  - Formally, the objective is given by

$$\operatorname*{argmin}_f \mathbb{E}_{\mathcal{T}_i}\ \mathcal{L}_i(\mathcal{T}_i; f_i)$$

  - Since we mostly use parametric models (or deep neural network), we will denote the parameter of meta-model and task-specific models as θ and ϕ_i, respectively

- **Few-shot classification**
  - Human can classify novel objects even though they see only a few samples
  - **Example:** Classify the breed of dogs (3-way 1-shot problem)

Pomeranian

Welsh Corgi

Siba Inu

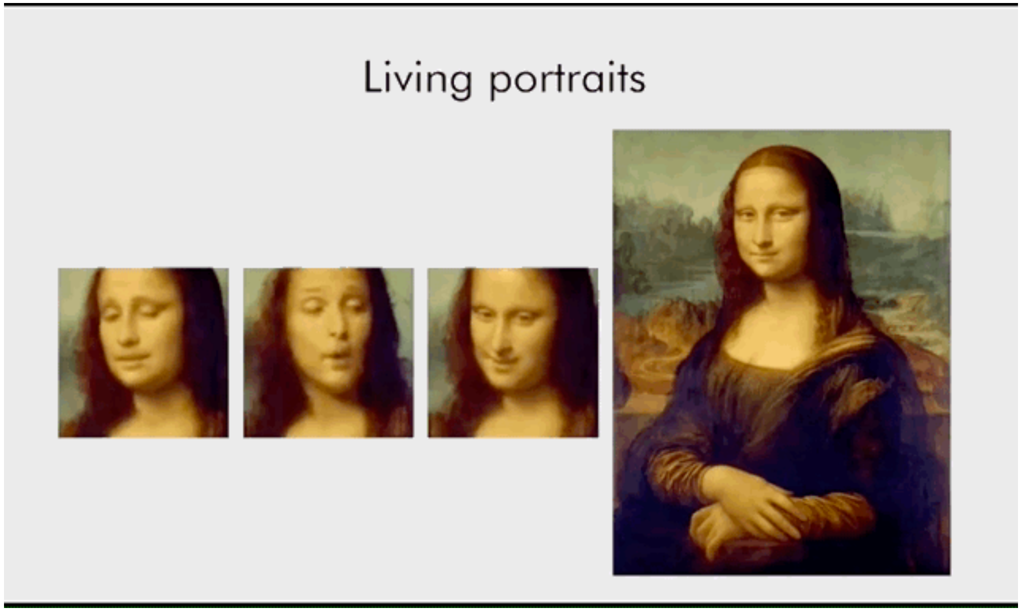
- **Q.** What is the breed of this dog?

- **Few-shot classification**
  - Human can classify novel objects even though they see only a few samples
  - Few-shot learning can be formulated as a **meta-learning** problem
    - **Task:** Given N classes of K samples each (i.e., N-way K-shot), predict the class of test samples (Each combination of N classes defines a task)
    - In this case, the meta model f learns a dog breed classifier f_{dog} from the given training images (and evaluated by test images)

- **Few-shot classification**
  - Classify novel instances with a few-shot of samples

- **Few-shot generation**
  - Generate novel instances of given samples

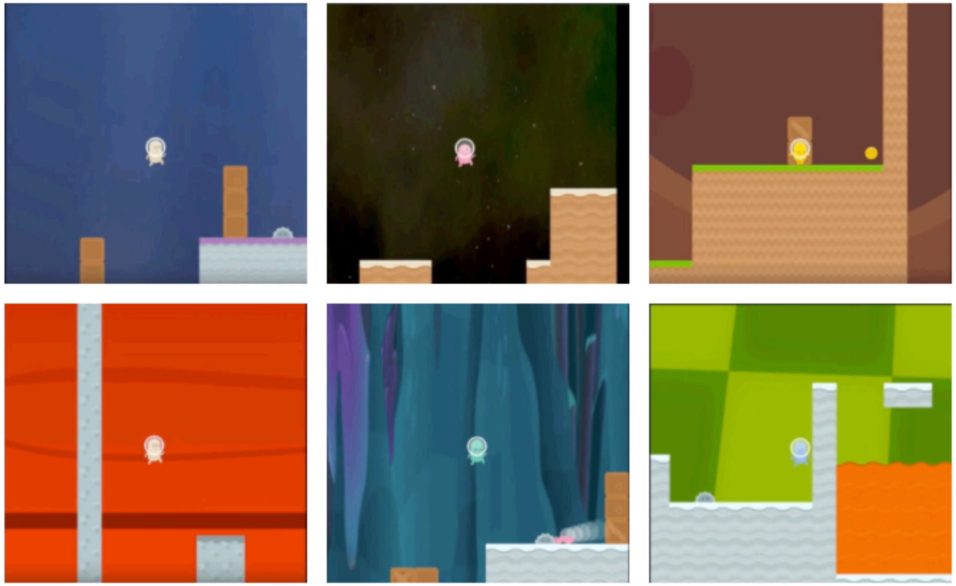
  - **Example:** Generate new emotions and angles of Mona Lisa (*unique* in the world!)



- **Few-shot classification**
  - Classify novel instances with a few-shot of samples

- **Few-shot generation**
  - Generate novel instances of given samples

- **Generalization of RL**
  - Generalize to novel environments



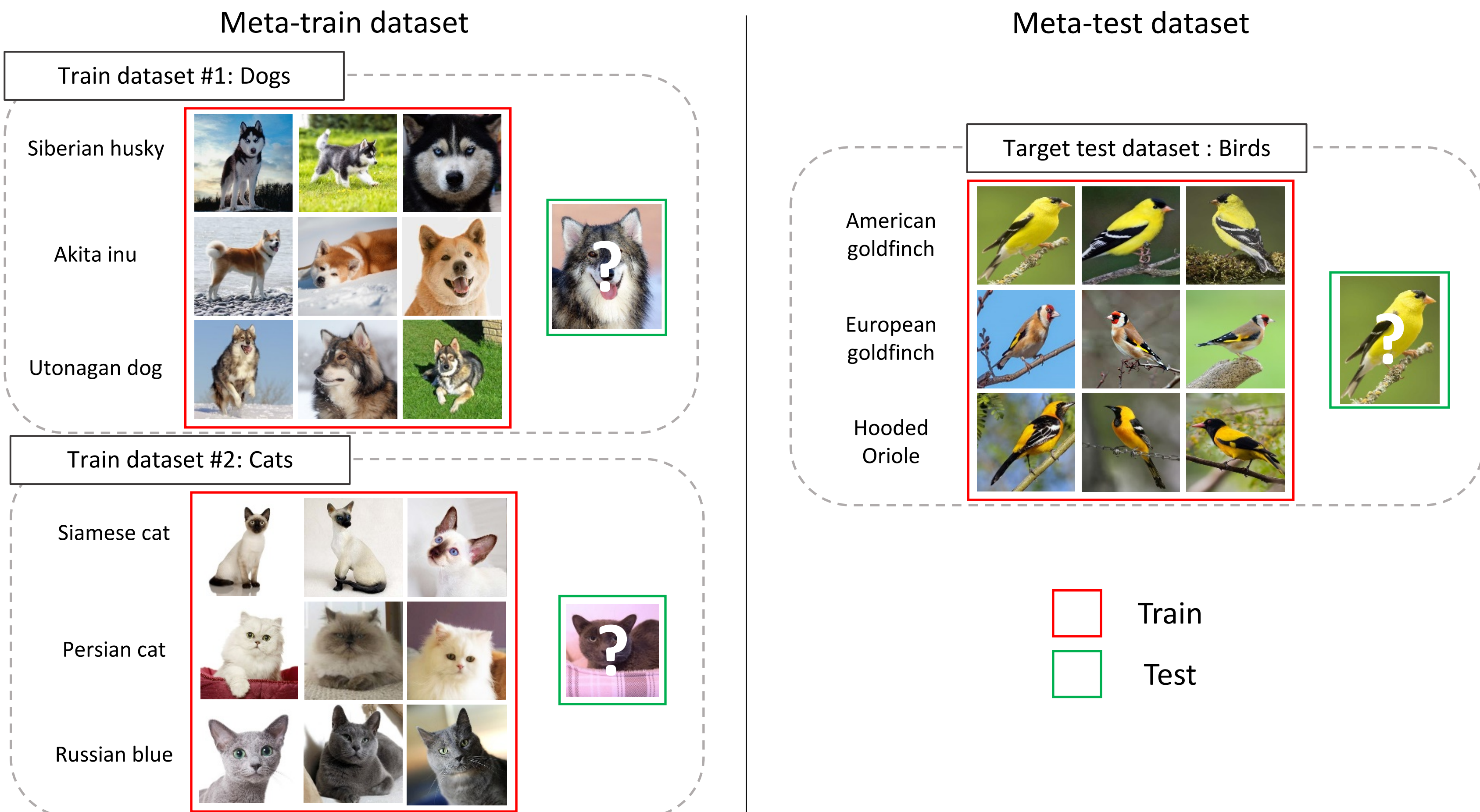
- **Few-shot classification**
  - Classify novel instances with a few-shot of samples

- **Few-shot generation**
  - Generate novel instances of given samples

- **Generalization of RL**
  - Generalize to novel environments

- **and LOTS of other applications**
  - Neural architecture search
  - Hyperparameter optimization
  - Loss function design
  - …and so on

- **Problem formulation**
  - To meta-learn a model, we need a meta-train dataset $\{(\mathcal{D}_i^{\text{train}}, \mathcal{D}_i^{\text{test}})\}$ consist of training and test datasets for each task $\mathcal{T}_i$
  - The performance of meta model is evaluated by a meta-test dataset



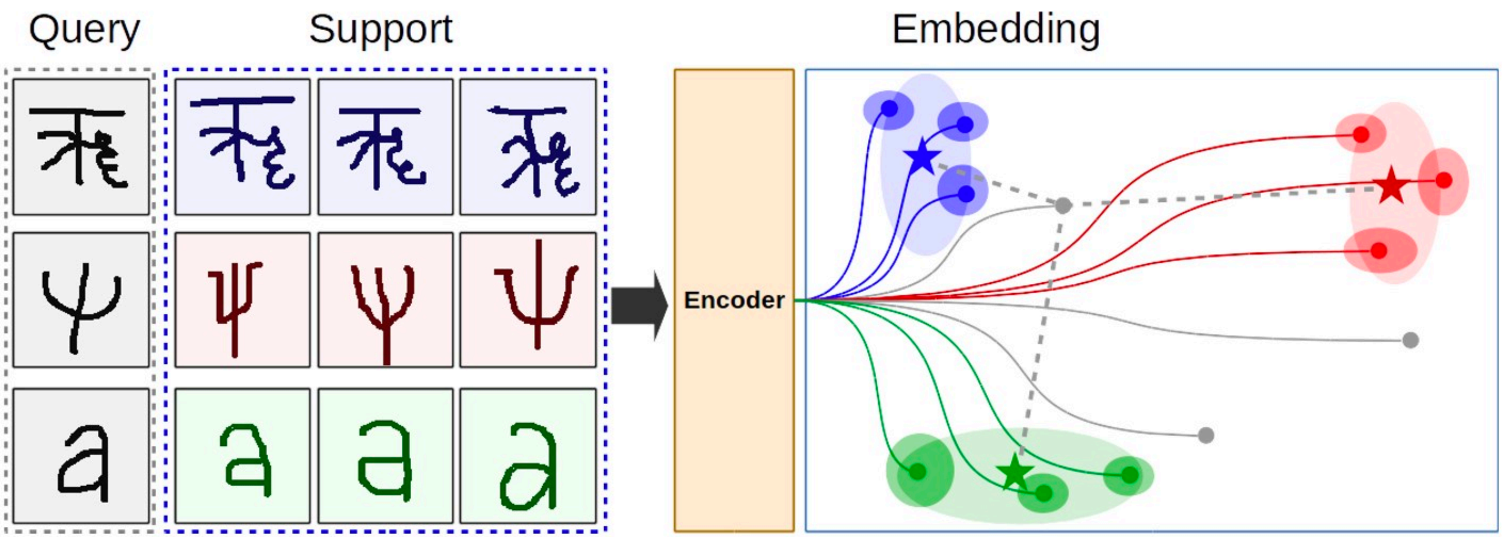
- **General recipe for meta-learning**
  - The core of meta-learning is how to learn a task-specific models for a given task
  - There are two common ways to ***learn*** the model from the dataset $\mathcal{D}_i^{\text{train}}$

- **Model-based meta-learning**
  - The meta-parameter θ is fixed, and the task is encoded to a context variable c_i
  - Namely, the task-specific function is given by $f(\cdot\,|\theta, c_i)$

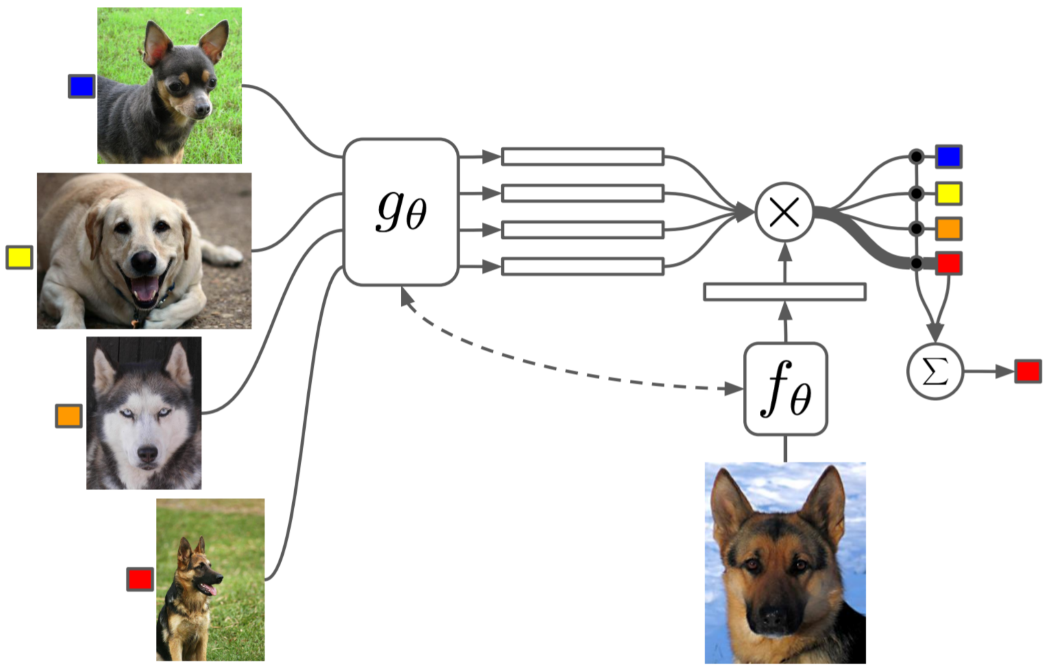
- **Optimization-based meta-learning**
  - Learn a parameter $\phi_i = g(\mathcal{D}_i^{\text{train}}; \theta)$ for each task $\mathcal{T}_i$
  - Namely, the task-specific function is given by $f(\cdot\,|\phi_i)$

  - Note that deep learning procedure can be decomposed into two steps:
    - How to set the initial parameter $\phi_i^{(0)}$
    - How to update the parameter $\phi_i^{(t)}$ to the better parameter $\phi_i^{(t+1)}$
  - The meta-learner θ will learn the initialization and/or update schemes

- **Metric-based meta-learning**
  - For a special type of meta-learning, **few-shot classification**, another common approach is to learn an embedding function and the corresponding metric
  - The embedding function maps similar samples to the similar embedding, and one can classify a novel sample by finding the nearest cluster



- **Matching Networks** [Vinyals et al. 16] propose to learn a shared embedding space over multiple subclassification problems.



[Vinyals et al. 16] Matching Networks for One Shot Learning, NIPS 2016

Matching network training objective:

$$\theta = \arg\max_{\theta} E_{L\sim T}\left[E_{S\sim L, B\sim L}\left[\sum_{(x,y)\in B} \log P_{\theta}\left(y|x, S\right)\right]\right]$$

Obtaining the optimal θ can be done via **episodic training**.

- First sample L (label set) from T, and use L to sample the support set S and a batch B.
- Then minimize the error predicting the labels in the batch B conditioned on the support set S.

Matching Networks

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

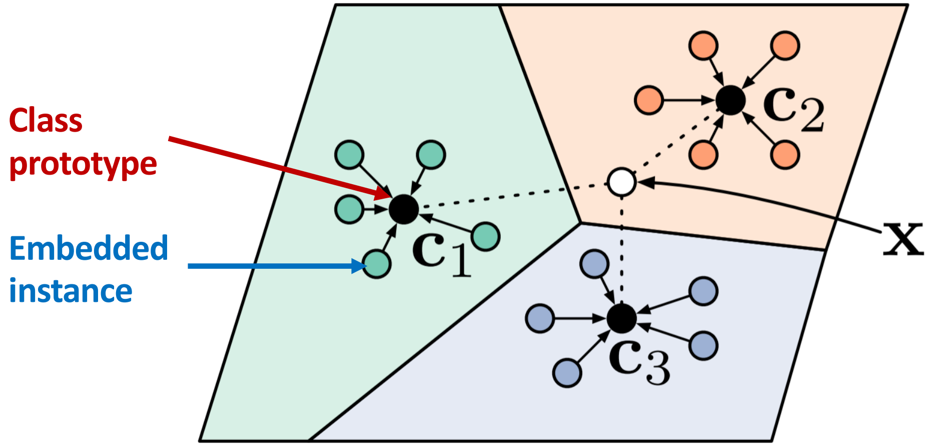
Model	Matching Fn	Fine Tune	5-way Acc 1-shot	5-way Acc 5-shot	20-way Acc 1-shot	20-way Acc 5-shot
PIXELS	Cosine	N	41.7%	63.2%	26.7%	42.6%
BASELINE CLASSIFIER	Cosine	N	80.0%	95.0%	69.5%	89.1%
BASELINE CLASSIFIER	Cosine	Y	82.3%	98.4%	70.6%	92.0%
BASELINE CLASSIFIER	Softmax	Y	86.0%	97.6%	72.9%	92.3%
MANN (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	**98.1%**	**98.9%**	**93.8%**	98.5%
MATCHING NETS (OURS)	Cosine	Y	97.9%	98.7%	93.5%	**98.7%**

Table 1: Results on the Omniglot dataset.

- Fine-tuning helped with baseline classifiers, but not in the case of Matching Networks.

[Vinyals et al. 16] Matching Networks for One Shot Learning, NIPS 2016

- **Prototypical Networks** [Snell et al. 17] use meta-learning to learn a metric space that minimizes the Euclidean distance between the prototypes and each training instance.



Target class

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

[Snell et al. 17] Prototypical Networks for Few-shot Learning, NIPS 2017

- Prototypical Networks are trained by minimizing the negative log-probability $J(\phi) = -\log p_\phi(y = k \mid \mathbf{x})$ via episodic training.

Input: Training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where each $y_i \in \{1, \ldots, K\}$. $\mathcal{D}_k$ denotes the subset of $\mathcal{D}$ containing all elements $(\mathbf{x}_i, y_i)$ such that $y_i = k$.

Output: The loss J for a randomly generated training episode.

```
V ← RANDOMSAMPLE({1, ..., K}, N_C)                         ▷ Select class indices for episode
for k in {1, ..., N_C} do
    S_k ← RANDOMSAMPLE(D_{V_k}, N_S)                       ▷ Select support examples
    Q_k ← RANDOMSAMPLE(D_{V_k} \ S_k, N_Q)                 ▷ Select query examples
    c_k ← (1/N_C) Σ_{(x_i, y_i) ∈ S_k} f_φ(x_i)            ▷ Compute prototype from support examples
end for
J ← 0                                                      ▷ Initialize loss
for k in {1, ..., N_C} do
    for (x, y) in Q_k do
        J ← J + (1/(N_C N_Q)) [ d(f_φ(x), c_k)) + log Σ_{k'} exp(−d(f_φ(x), c_{k'})) ]   ▷ Update loss
    end for
end for
```

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

$$J \leftarrow J + \frac{1}{N_C N_Q}\left[d(f_\phi(\mathbf{x}), \mathbf{c}_k)) + \log \sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'}))\right]$$

[Snell et al. 17] Prototypical Networks for Few-shot Learning, NIPS 2017

- Prototypical Networks outperform Matching Networks and MAML on few-shot classification tasks.

Omniglot

Model	Dist.	Fine Tune	5-way Acc. 1-shot	5-way Acc. 5-shot	20-way Acc. 1-shot	20-way Acc. 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%	**99.9%**	95.8%	**98.9%**
PROTOTYPICAL NETWORKS (OURS)	Euclid.	N	**98.8%**	99.7%	**96.0%**	**98.9%**

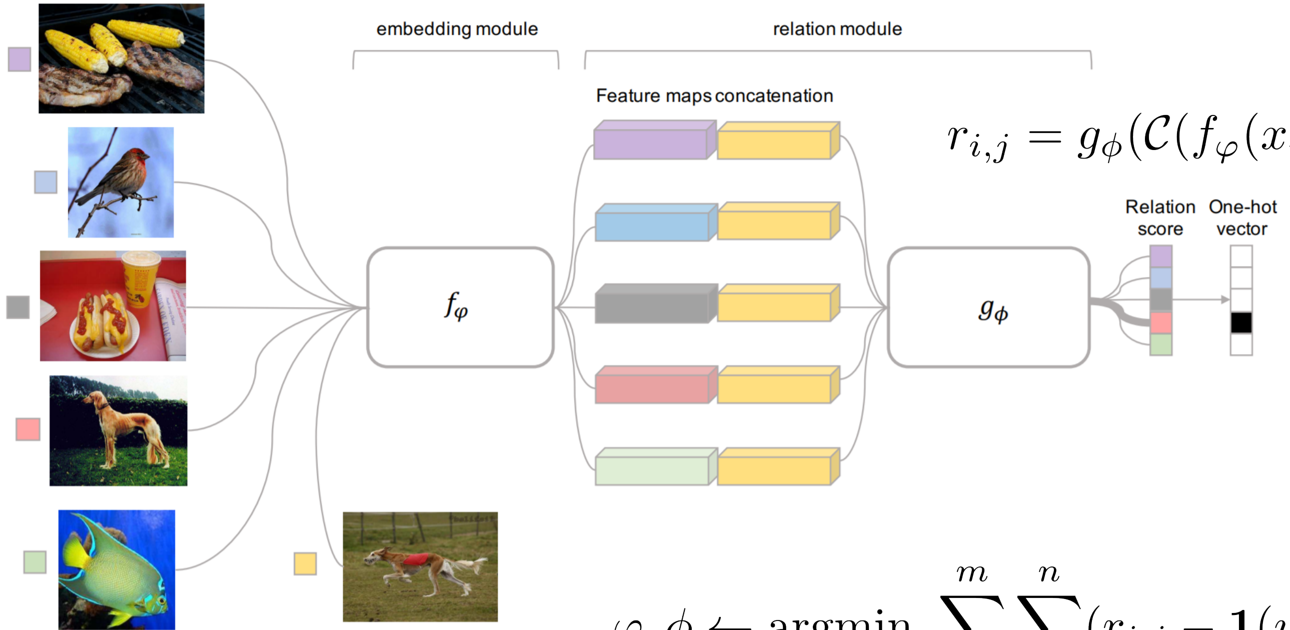
miniImageNet

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

[Snell et al. 17] Prototypical Networks for Few-shot Learning, NIPS 2017

Relation Networks

- **Relation Networks** [Sung et al. 18] learns to learn a deep metric space by learning to minimize the relation scores between the query and the support samples.



$$r_{i,j} = g_\phi(\mathcal{C}(f_\varphi(x_i), f_\varphi(x_j)))$$

$$\varphi, \phi \leftarrow \underset{\varphi,\phi}{\operatorname{argmin}} \sum_{i=1}^{m} \sum_{j=1}^{n} (r_{i,j} - \mathbf{1}(y_i == y_j))^2$$

[Sung et al. 18] Learning to Compare: Relation Networks for Few-shot Learning, CVPR 2018

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

Omniglot

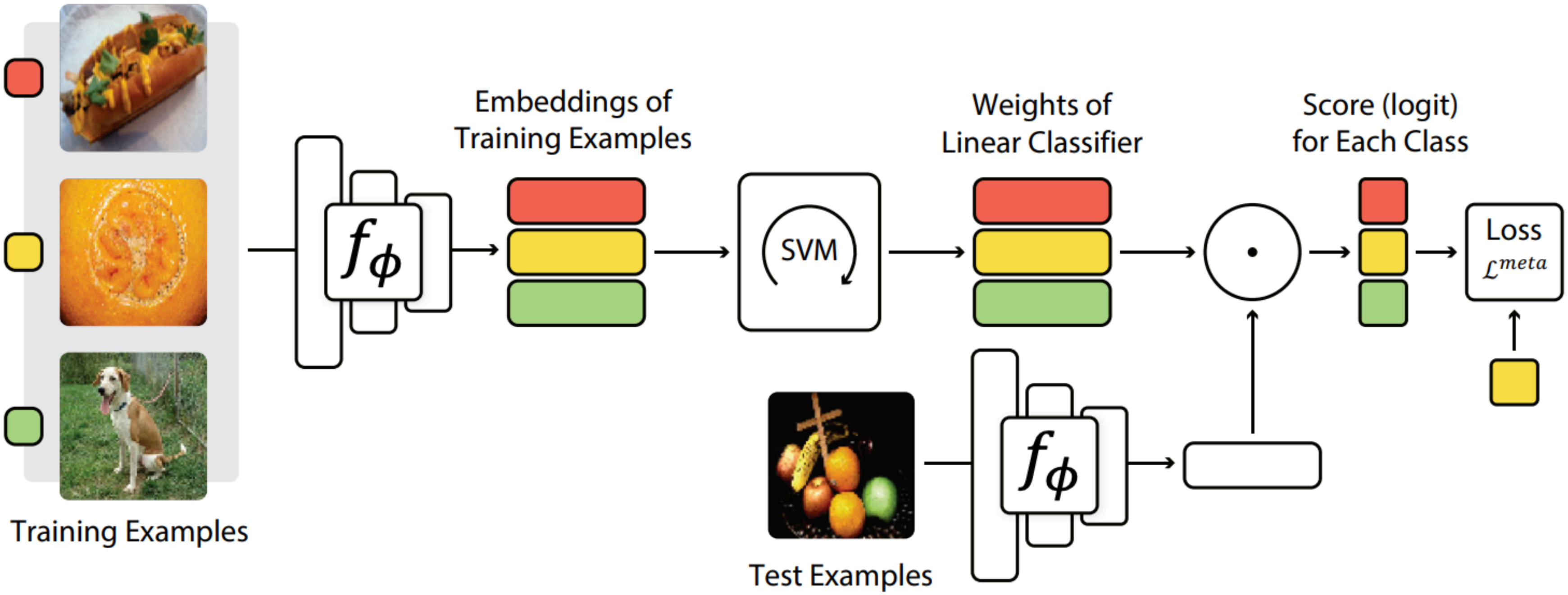
Model	Fine Tune	5-way Acc.		20-way Acc.	
		1-shot	5-shot	1-shot	5-shot
MANN [32]	N	82.8%	94.9%	-	-
CONVOLUTIONAL SIAMESE NETS [20]	N	96.7%	98.4%	88.0%	96.5%
CONVOLUTIONAL SIAMESE NETS [20]	Y	97.3%	98.4%	88.1%	97.0%
MATCHING NETS [39]	N	98.1%	98.9%	93.8%	98.5%
MATCHING NETS [39]	Y	97.9%	98.7%	93.5%	98.7%
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%

[Sung et al. 18] Learning to Compare: Relation Networks for Few-shot Learning, CVPR 2018

- **MetaOptNet** [Lee et al. 19] uses more complex classifiers (e.g., SVM) instead of the naïve nearest neighbor classifier, upon the learned embedding



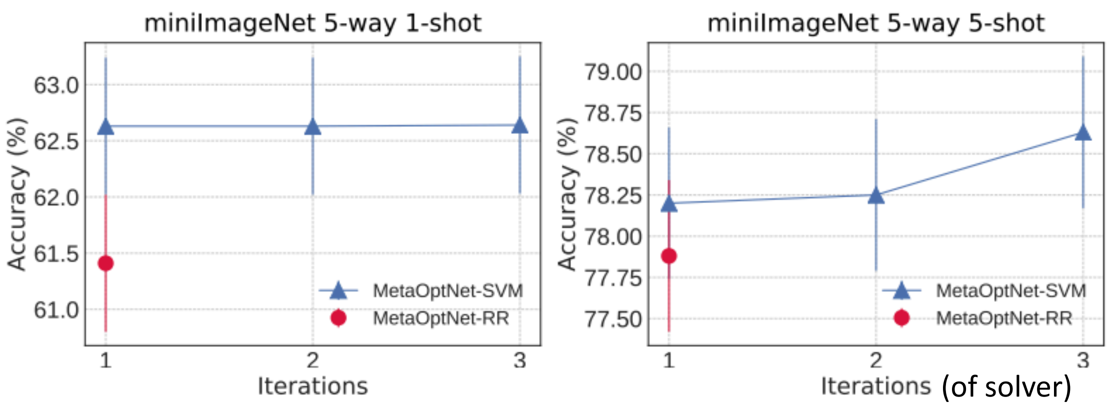
- Here, the classifier is defined by a closed form solution of some quadratic programming (QP) problem

$$\theta = \mathcal{A}(\mathcal{D}^{train}; \phi) = \underset{\{\boldsymbol{w}_k\}}{\arg\min} \min_{\{\xi_i\}} \frac{1}{2} \sum_k ||\boldsymbol{w}_k||_2^2 + C \sum_n \xi_n$$

[Lee et al. 19] Meta-Learning with Differentiable Convex Optimization, CVPR 2019

- MetaOptNet with ridge regression (RR) and support vector machine (SVM) shows better results than naïve prototypical network

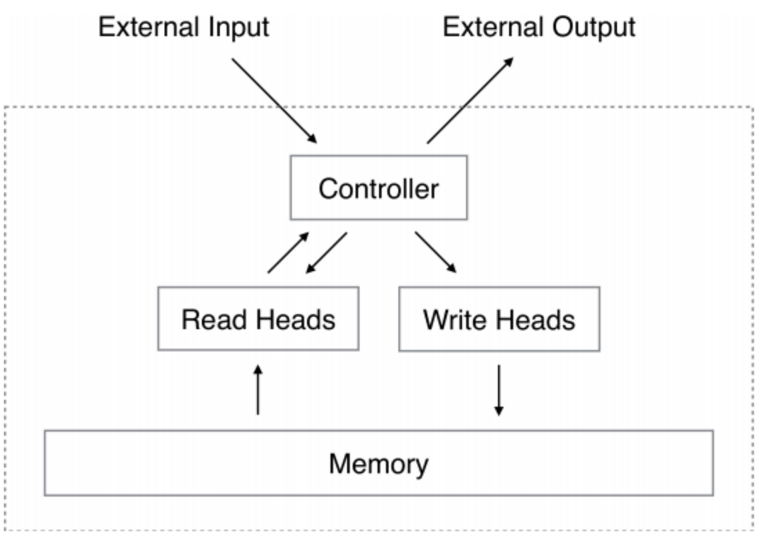
model	miniImageNet 5-way 1-shot acc. (%)	miniImageNet 5-way 1-shot time (ms)	miniImageNet 5-way 5-shot acc. (%)	miniImageNet 5-way 5-shot time (ms)	tieredImageNet 5-way 1-shot acc. (%)	tieredImageNet 5-way 1-shot time (ms)	tieredImageNet 5-way 5-shot acc. (%)	tieredImageNet 5-way 5-shot time (ms)
4-layer conv (feature dimension=1600)								
Prototypical Networks [17, 28]	$53.47_{\pm 0.63}$	$6_{\pm 0.01}$	$70.68_{\pm 0.49}$	$7_{\pm 0.02}$	$54.28_{\pm 0.67}$	$6_{\pm 0.03}$	$71.42_{\pm 0.61}$	$7_{\pm 0.02}$
MetaOptNet-RR (ours)	$53.23_{\pm 0.59}$	$20_{\pm 0.03}$	$69.51_{\pm 0.48}$	$27_{\pm 0.05}$	$54.63_{\pm 0.67}$	$21_{\pm 0.05}$	$72.11_{\pm 0.59}$	$28_{\pm 0.06}$
MetaOptNet-SVM (ours)	$52.87_{\pm 0.57}$	$28_{\pm 0.02}$	$68.76_{\pm 0.48}$	$37_{\pm 0.05}$	$54.71_{\pm 0.67}$	$28_{\pm 0.07}$	$71.79_{\pm 0.59}$	$38_{\pm 0.08}$
ResNet-12 (feature dimension=16000)								
Prototypical Networks [17, 28]	$59.25_{\pm 0.64}$	$60_{\pm 17}$	$75.60_{\pm 0.48}$	$66_{\pm 17}$	$61.74_{\pm 0.77}$	$61_{\pm 17}$	$80.00_{\pm 0.55}$	$66_{\pm 18}$
MetaOptNet-RR (ours)	$61.41_{\pm 0.61}$	$68_{\pm 17}$	**77.88**$_{\pm 0.46}$	$75_{\pm 17}$	**65.36**$_{\pm 0.71}$	$69_{\pm 17}$	**81.34**$_{\pm 0.52}$	$77_{\pm 17}$
MetaOptNet-SVM (ours)	**62.64**$_{\pm 0.61}$	$78_{\pm 17}$	**78.63**$_{\pm 0.46}$	$89_{\pm 17}$	**65.99**$_{\pm 0.72}$	$78_{\pm 17}$	**81.56**$_{\pm 0.53}$	$90_{\pm 17}$



[Lee et al. 19] Meta-Learning with Differentiable Convex Optimization, CVPR 2019

Table of Contents

- [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.
- [Santoro et al. 16] proposed memory-augmented neural network (MANN) to rapidly assimilate new data, and to make accurate predictions with few samples.

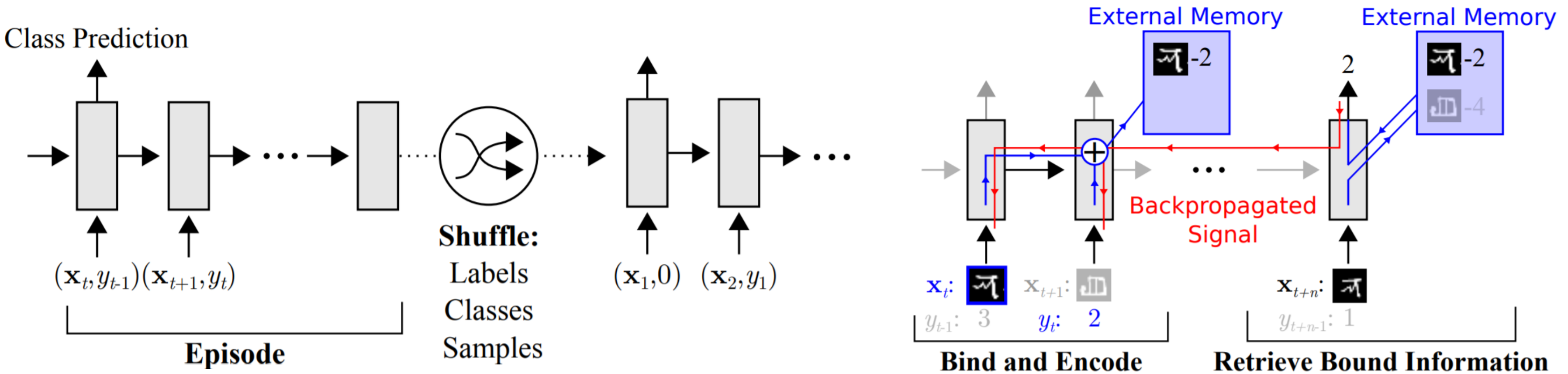


Neural Turing Machine

[Graves et al. 14] Neural Turing Machines, 2014

Meta-Learning with MANN

- 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 time to time to prevent weights from binding to sample-class binding.

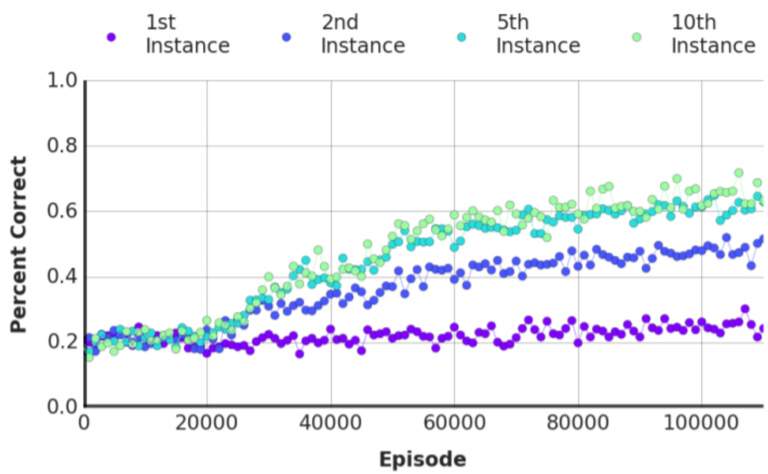


- This method enables to learn a generic scheme to bind representations to their appropriate labels regardless of the actual contents of data representations or labels.

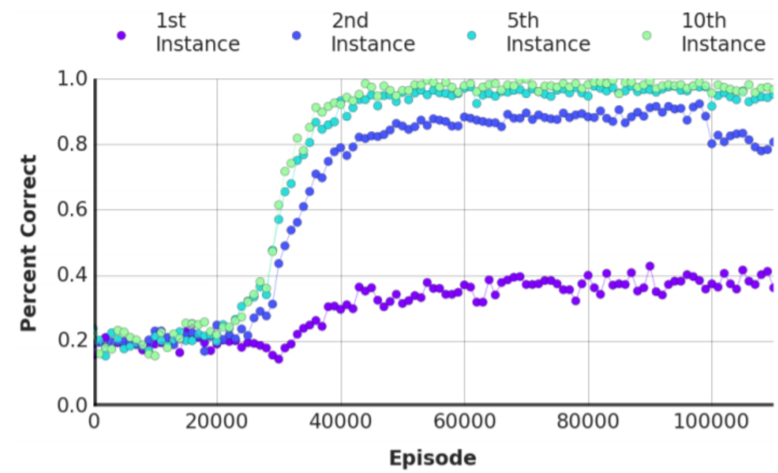
[Santoro et al. 16] Meta-Learning with Memory-Augmented Neural Networks, ICML 2016

Meta-Learning with MANN

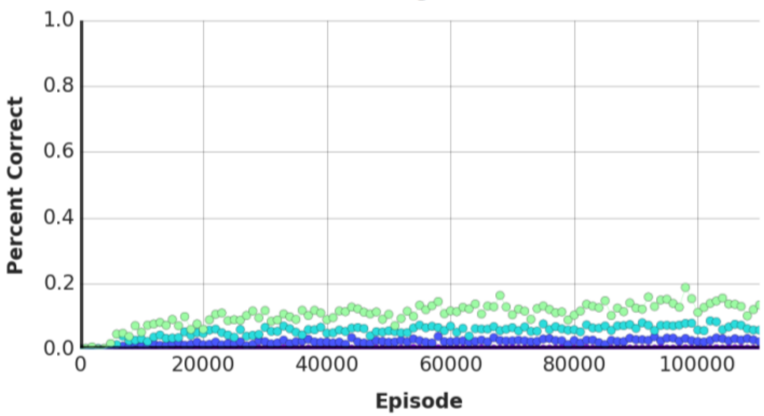
- MANN significantly outperforms LSTM (which has internal memory) for few-shot classification on Omniglot dataset.



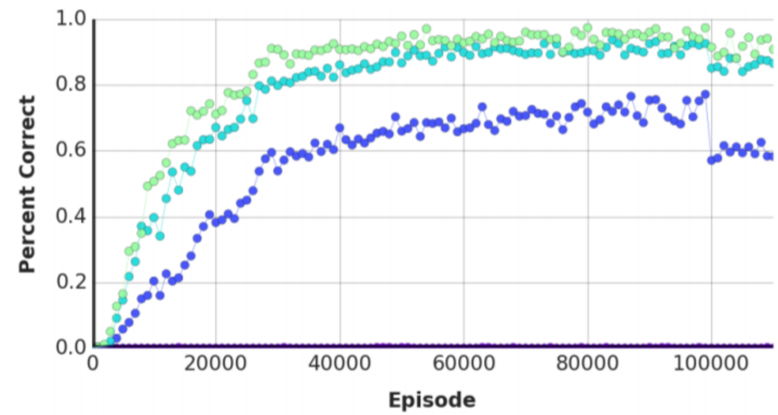
(a) LSTM, five random classes/episode, one-hot vector labels



(b) MANN, five random classes/episode, one-hot vector labels



(c) LSTM, fifteen classes/episode, five-character string labels



(d) MANN, fifteen classes/episode, five-character string labels

[Santoro et al. 16] Meta-Learning with Memory-Augmented Neural Networks, ICML 2016

- 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.

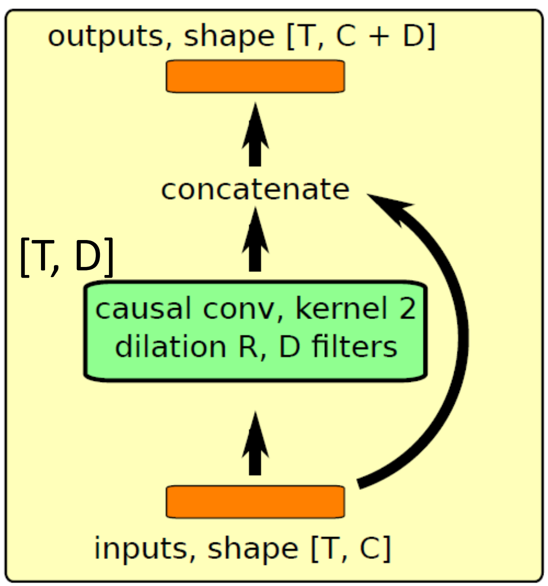
- These two components complement each other: while the former provide **high-bandwidth access** at the expense of **finite context size**, the latter provide **pinpoint access** over **an infinitely large context**.

[Mishra et al. 18] A Simple Neural Attentive Meta-Learner, ICLR 2018

- 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.

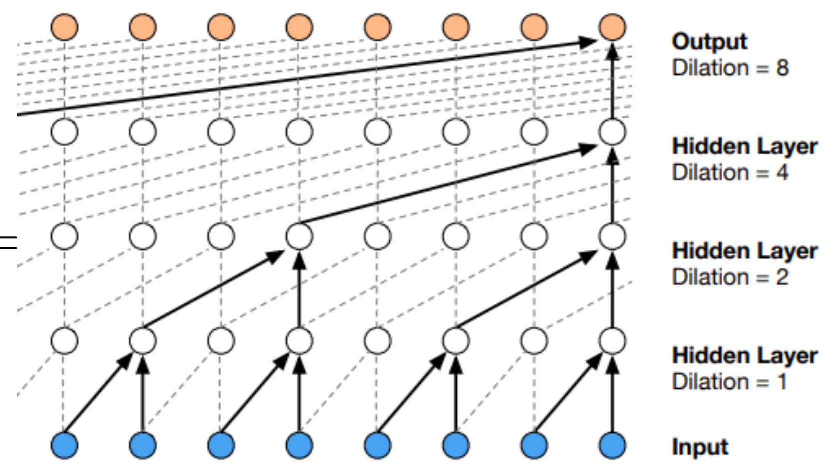
function TCBLOCK(inputs, sequence length T, number of filters D):
    for i in $1, \ldots, \lceil \log_2 T \rceil$ **do**
        inputs = DenseBlock(inputs, 2^i, D)
    return inputs

TC Block =



Dense Block

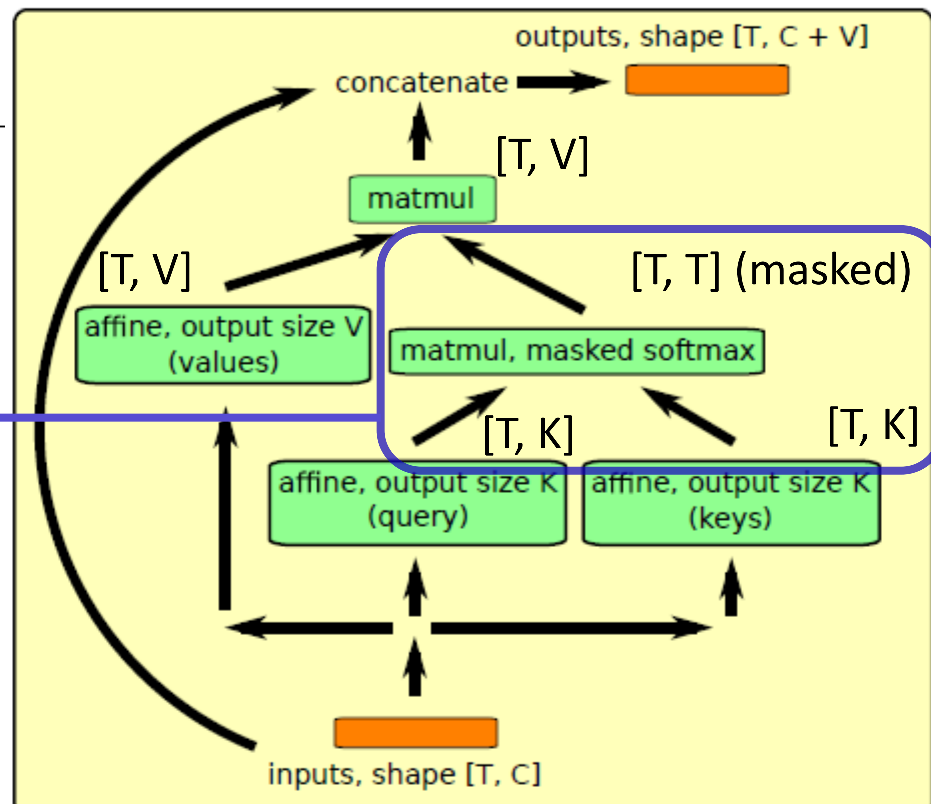
$\times \log T =$



[Mishra et al. 18] A Simple Neural Attentive Meta-Learner, ICLR 2018

- 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.

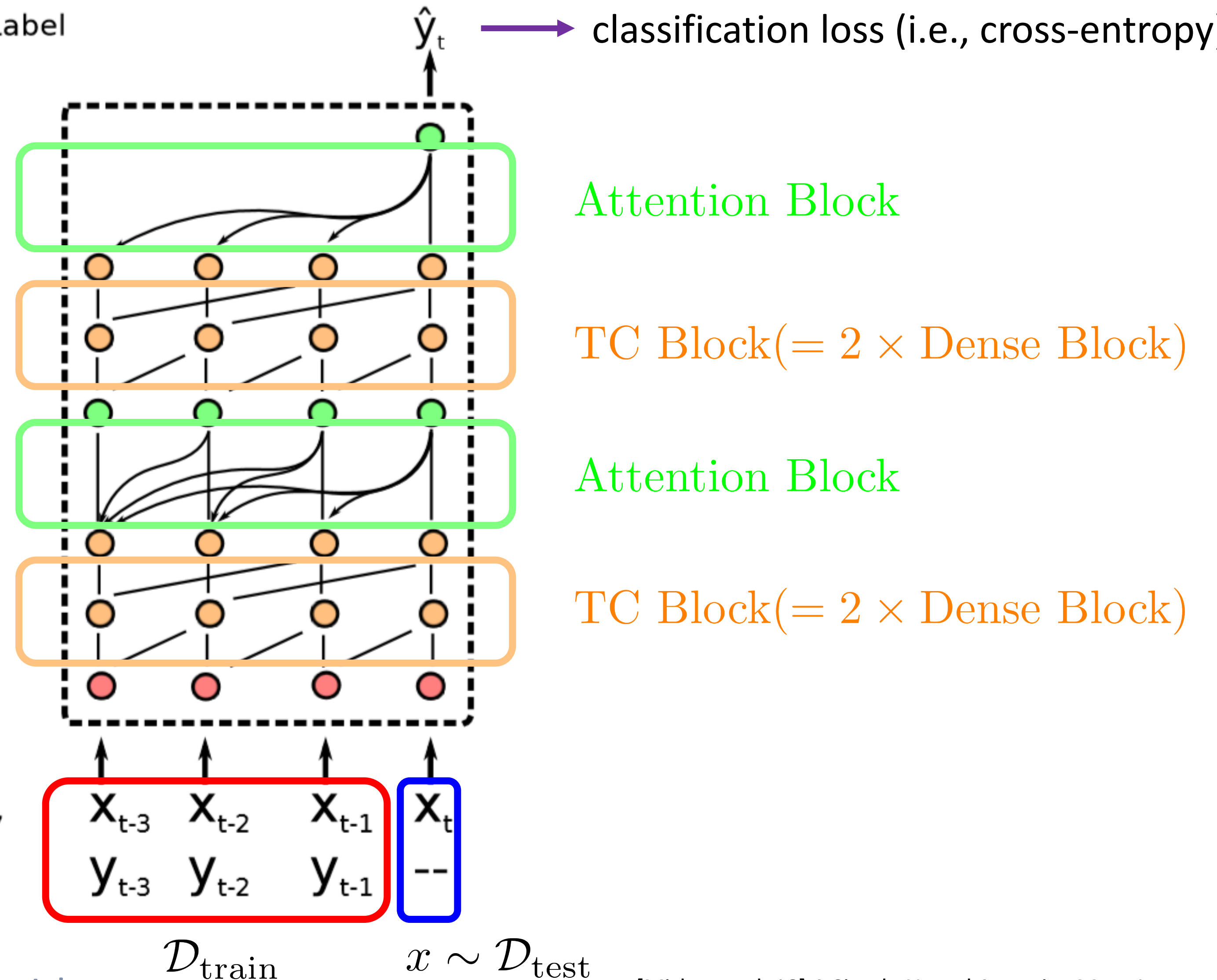
```
function AttentionBlock(inputs, key size K, value size V):
    keys, query = affine(inputs, K), affine(inputs, K)
    logits = matmul(query, transpose(keys))
    probs = CausallyMaskedSoftmax(logits / √K)
    values = affine(inputs, V)
    read = matmul(probs, values)
    return concat(inputs, read)
```



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

- Overview of the SNAIL for supervised learning:

Predicted Label



$\hat{y}_t$ → classification loss (i.e., cross-entropy)

[Mishra et al. 18] A Simple Neural Attentive Meta-Learner, ICLR 2018

Simple Neural AttentIve meta-Learner (SNAIL)

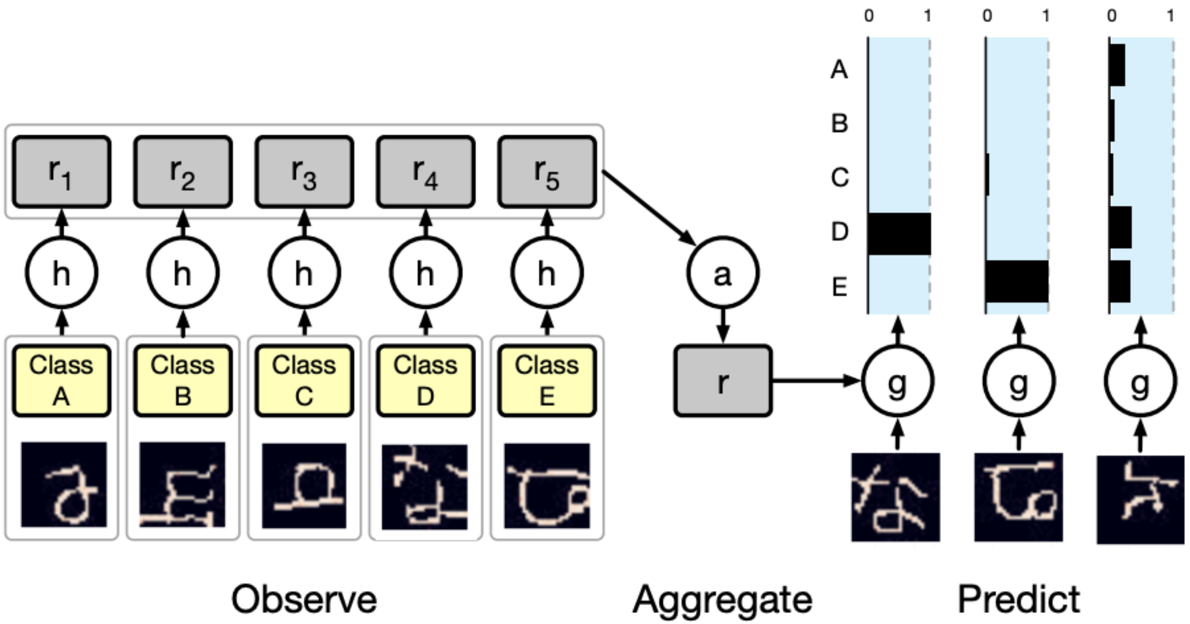
- SNAIL outperforms state-of-the-art methods in few-shot classification tasks that are extensively hand-designed, and/or domain-specific (e.g., Matching networks [Vinyals et al. 16]).
- It significantly exceeds the performance of methods such as MANN that are similarly simple and generic.

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)	**98.7% $\pm$ 0.4%**	**99.9% $\pm$ 0.3%**	95.8% $\pm$ 0.3%	98.9% $\pm$ 0.2%
Snell et al. (2017)	97.4%	99.3%	96.0%	98.9%
Munkhdalai & Yu (2017)	**98.9%**	–	97.0%	–
SNAIL, Ours	**99.07% $\pm$ 0.16%**	**99.78% $\pm$ 0.09%**	**97.64% $\pm$ 0.30%**	**99.36% $\pm$ 0.18%**

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

[Mishra et al. 18] A Simple Neural Attentive Meta-Learner, ICLR 2018

- **Conditional Neural Process (CNP)** [Garnelo et al. 18] extracts the context variable of task with set encoder, and predicts target under the context

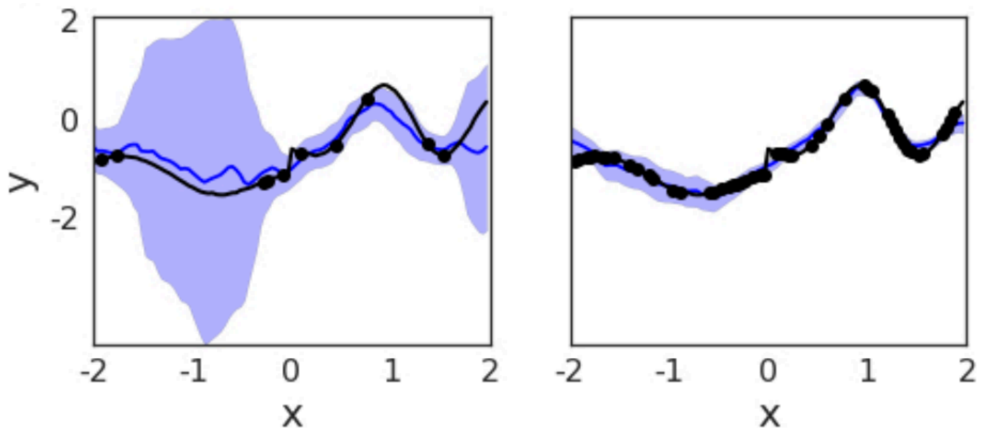


- Given observation O_N, model predicts outputs for both observed and unobserved samples, and trained to maximize the likelihood

$$\mathcal{L}(\theta) = -\mathbb{E}_{f\sim P}\left[\mathbb{E}_N\left[\log Q_\theta(\{y_i\}_{i=0}^{n-1}|O_N, \{x_i\}_{i=0}^{n-1})\right]\right]$$

[Garnelo et al. 18] Conditional Neural Process, ICML 2018

- Conditional Neural Process (CNP) behaves like a neural version of Gaussian process, e.g., it can predict uncertainty of outputs



- CNP is also computationally efficient as the input information is amortized to a single context variable, hence it has linear complexity

	5-way Acc		20-way Acc		Runtime
	1-shot	5-shot	1-shot	5-shot	
MANN	82.8%	94.9%	-	-	$\mathcal{O}(nm)$
MN	**98.1%**	**98.9%**	**93.8%**	**98.5%**	$\mathcal{O}(nm)$
CNP	95.3%	98.5%	89.9%	96.8%	$\mathcal{O}(n+m)$

Omniglot classification

[Garnelo et al. 18] Conditional Neural Process, ICML 2018

Table of Contents

1. **Introduction**
    - What is meta-learning?
    - Applications of meta-learning
    - Overview of common approaches

2. **Approaches to Meta-learning**
    - Metric-based meta-learning
    - Model-based meta-learning
    - Optimization-based meta-learning
        - Learning model initialization
        - Learning optimizers

- **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

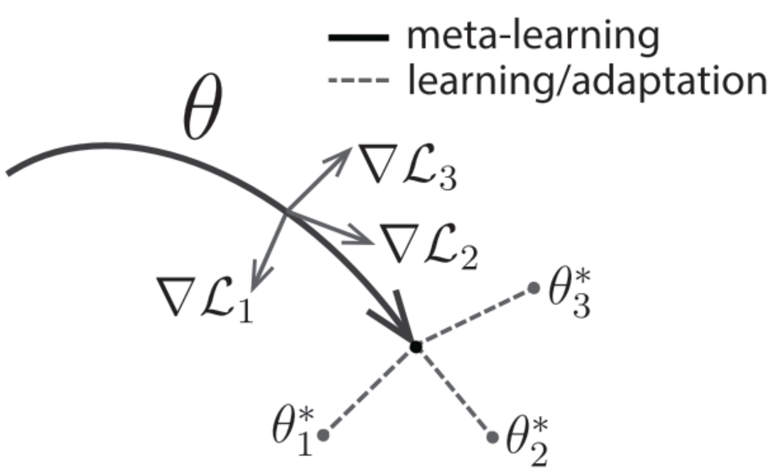
$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

(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**

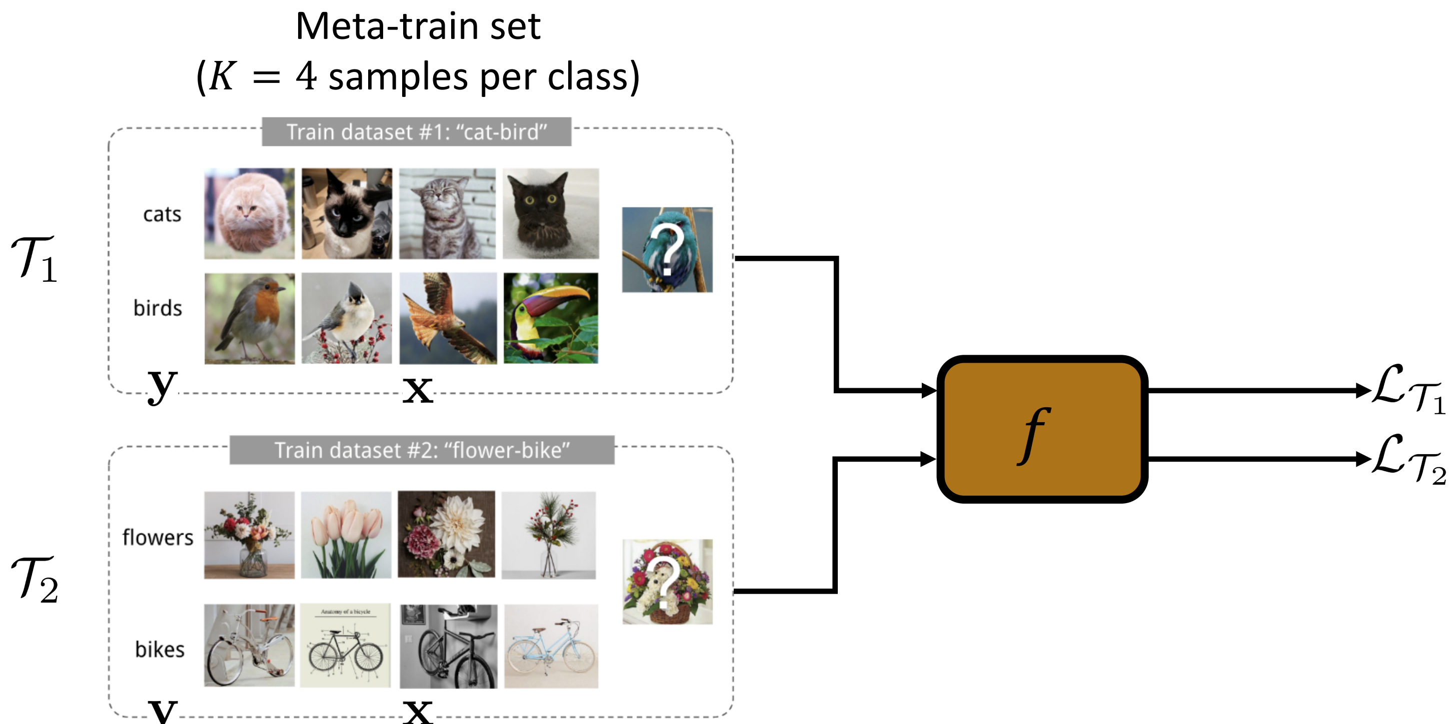
- Key idea
  - Train over many tasks, to learn parameter θ that transfers well
  - Use objective that **encourage** θ to **fast adapt** when fine-tuned with small data
  - Assumption: some representations are more transferrable than others

- 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



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 $\mathcal{T}_i$

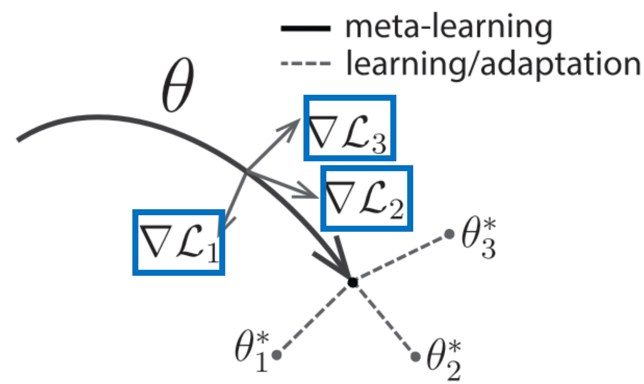


- Consider a model f_θ parameterized with θ
- Inner-loop
  - Adapting model to a new task $\mathcal{T}_i$

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

Where α is learning rate,

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

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

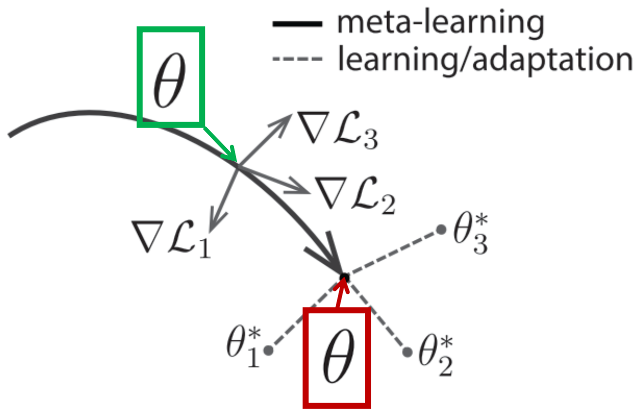
Where β is meta-learning rate

- 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:

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

Where β is meta-learning rate

Meta-Gradients of MAML

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

Task-specificly optimized parameters

Meta-learned initial model parameters

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

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

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

First Order Approximation of MAML

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

Task-specificly optimized parameters

Meta-learned initial model parameters

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

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

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

- Use 1st order approximation

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

  - Ignore 2nd 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** $\mathcal{L}_{\mathtt{mo}}$: 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 $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
  end for
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while

Inner loop

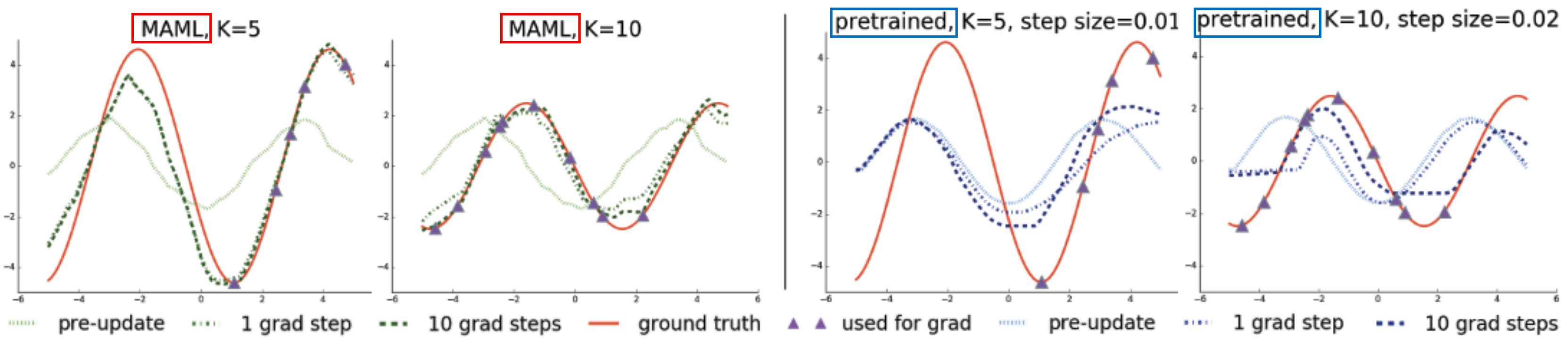
Outer loop

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

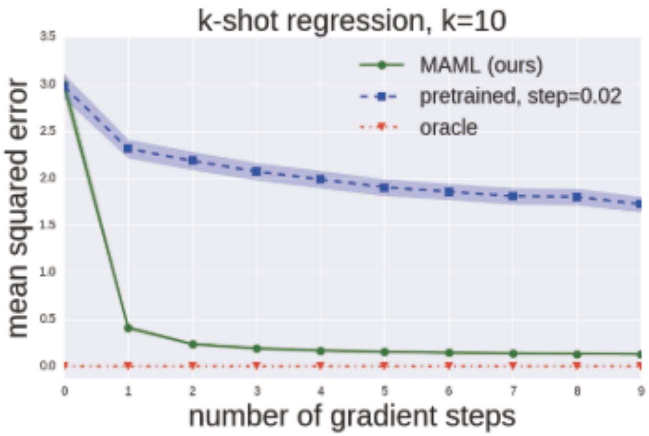
- Few-shot regression experiments
  - Regress the sine wave $y = A\sin(wx)$
  - Where $A \in [0.1, 5.0]$, $w \in [0, \pi]$, $x \in [-5, 5]$ are randomly sampled
  - MAML with one gradient update inner loop
  - Evaluate performance by fine-tuning the model
    - On K-samples, compared with simply pre-trained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

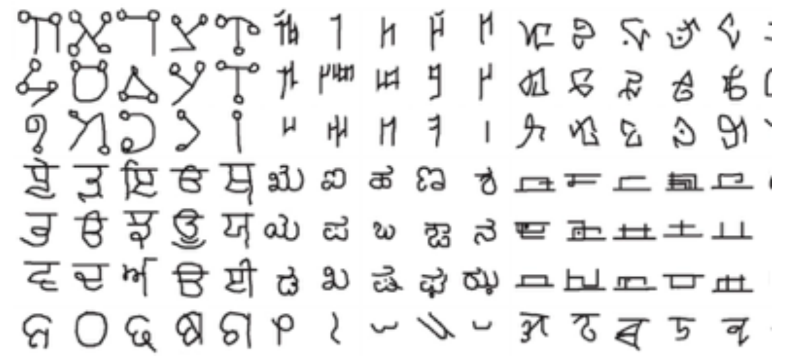
- Adapt much faster with small number of samples (purple triangle below)
  - **Continue to improve** with additional gradient step
    - Not overfitted to θ that only improves after one step
    - Learn initialization that amenable to fast adaptation



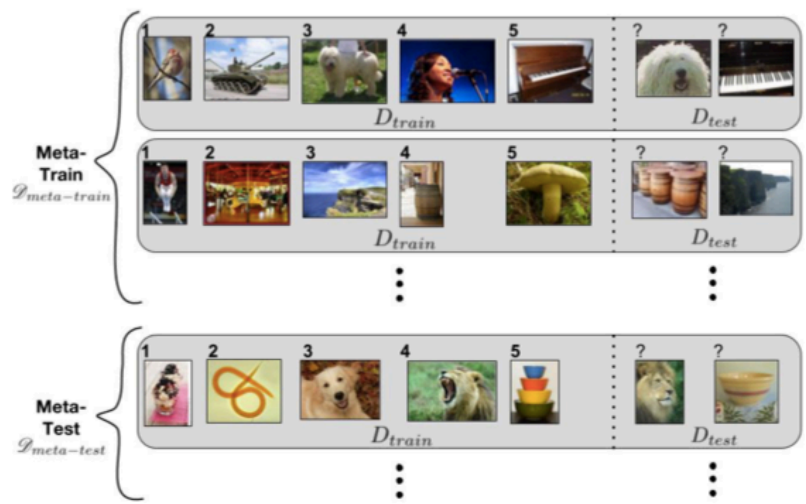
Experiments on Few-Shot Learning Tasks

- 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;

- 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 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

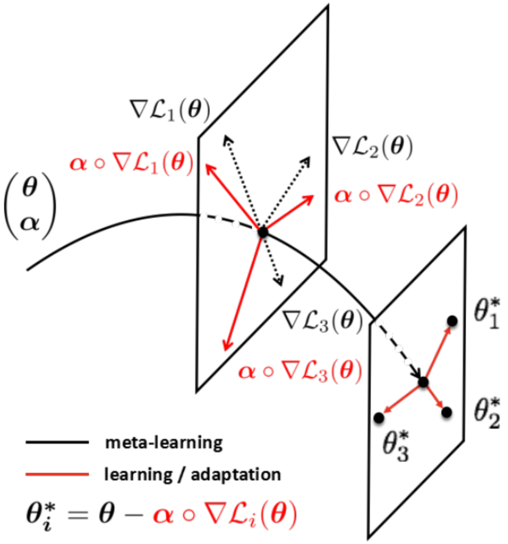
- Many recent works on meta-learning based on MAML
  - Learning the learning rate as well [Li, et. al., 2017]
  - First-order approximation of MAML [Nichol, et. al., 2018]
  - Probabilistic MAML [Finn, et. al., 2018]
  - Visual imitation learning [Finn, et. al., 2017]

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

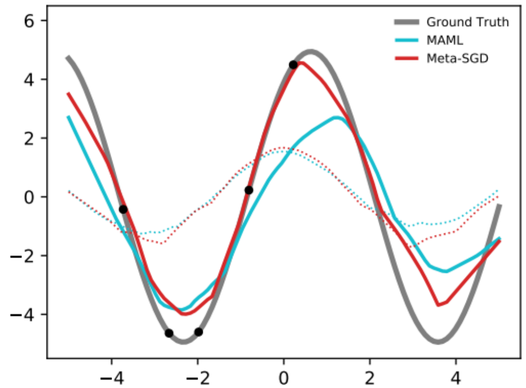
- Inner loop computation becomes: $\boldsymbol{\theta}' = \boldsymbol{\theta} - \boldsymbol{\alpha} \circ \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\boldsymbol{\theta}})$
  - Where $\boldsymbol{\alpha}$ is a vector of learning rates



* 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



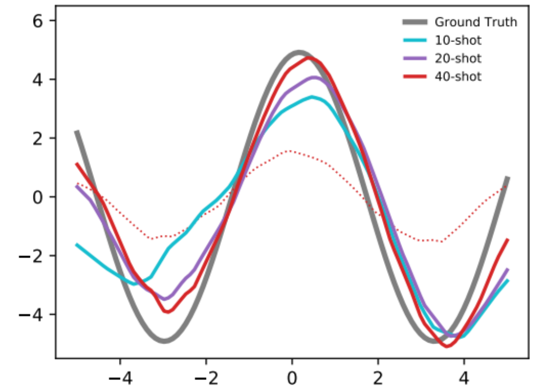


Figure 3: **Left:** Meta-SGD vs MAML on 5-shot regression. Both initialization (dotted) and result after one-step adaptation (solid) are shown. **Right:** Meta-SGD (10-shot meta-training) performs better with more training examples in meta-testing.

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

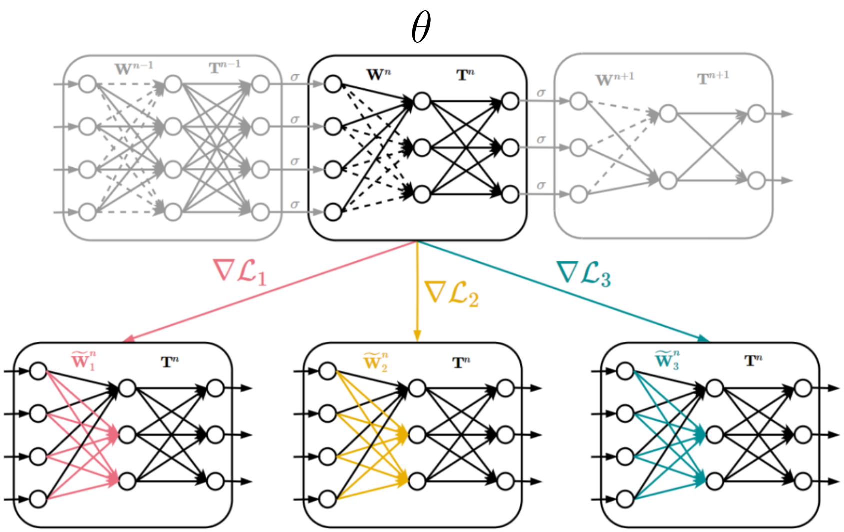
	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Matching Nets	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$	$17.31 \pm 0.22\%$	$22.69 \pm 0.20\%$
Meta-LSTM	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$	$16.70 \pm 0.23\%$	$26.06 \pm 0.25\%$
MAML	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$	$16.49 \pm 0.58\%$	$19.29 \pm 0.29\%$
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*

- MT-NET [Choi et al. 18] proposes a MAML variant that chooses a subset of weights to fine-tune.



MAML

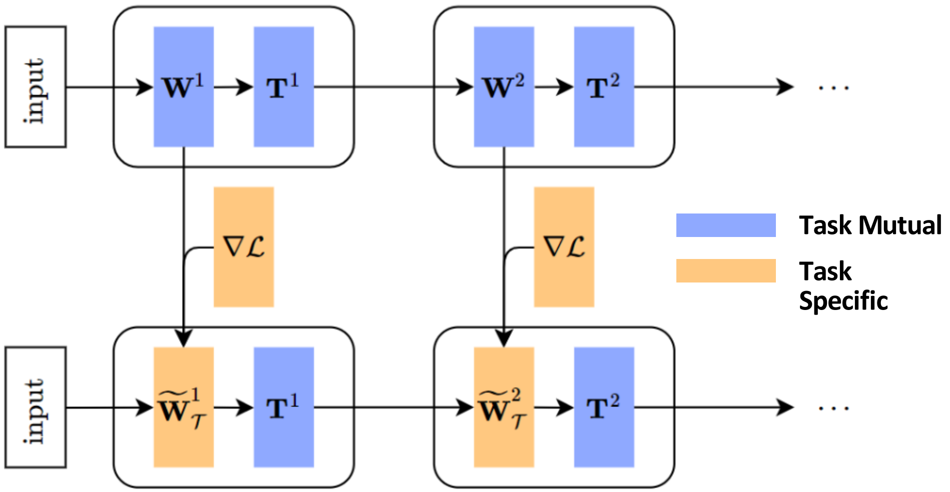
$$\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

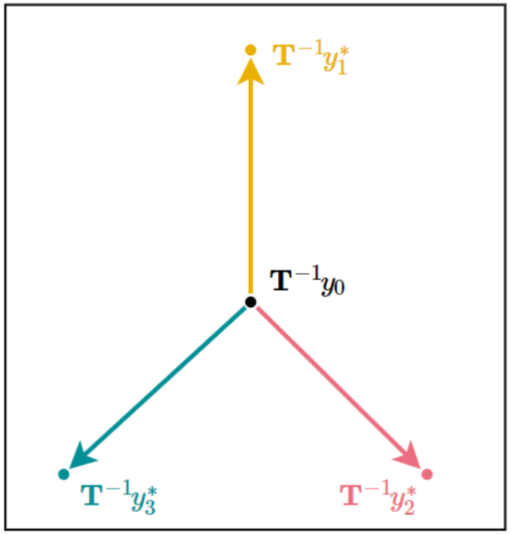
MT-NET

$$\widetilde{W}_{\mathcal{T}} \leftarrow W - \alpha\nabla_W\mathcal{L}(\theta_W, \theta_T, \mathcal{D}_{\mathcal{T},\text{train}})$$

- A model f_θ consists of L cells, where each cell is parameterized as TW.
- The meta-learner specifies weights to be changed(dotted line) over initial weights(black) as chosen by task-specific learners(colored).

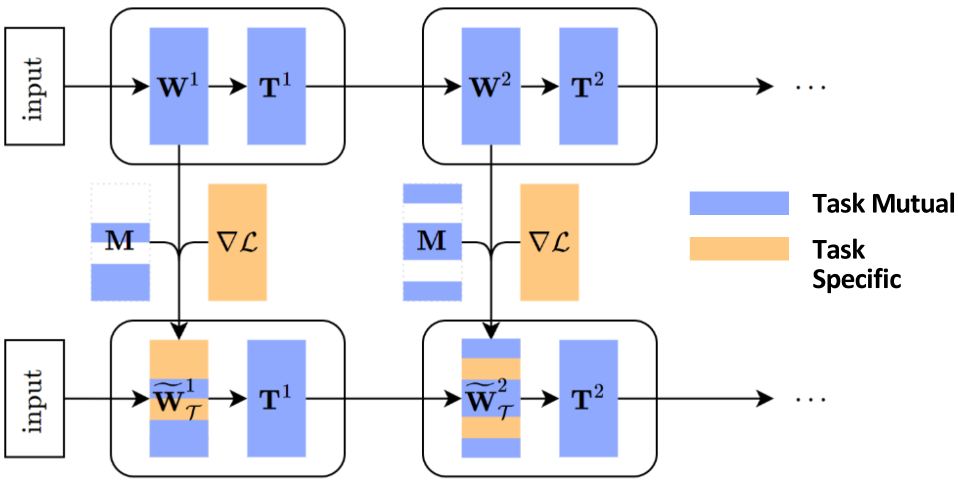
- A model f_θ consists of L cells, where each cell is parameterized as TW.

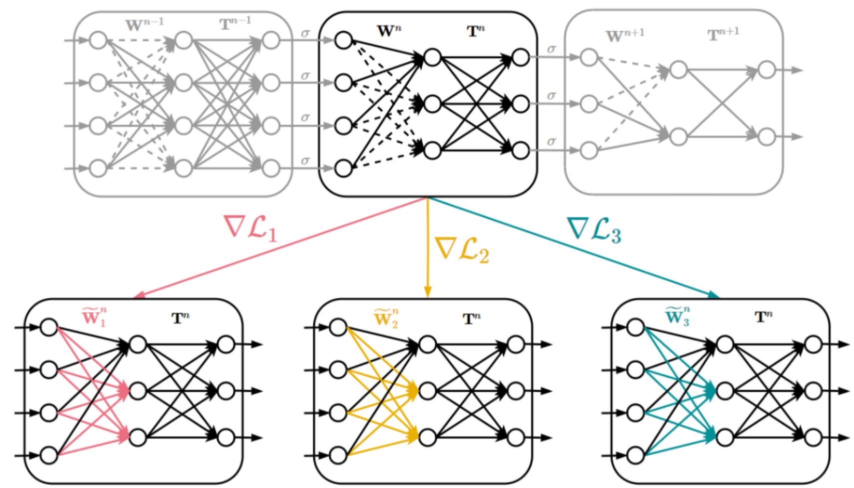




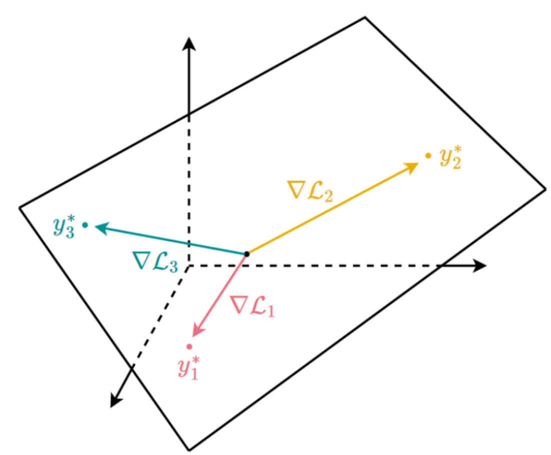
- T matrix learns a metric in activation space so that task specific weights W can preserve task identity.

- By adding binary mask, which selects weights to be updated, MT-NET chooses subspace that contributes to generalization.

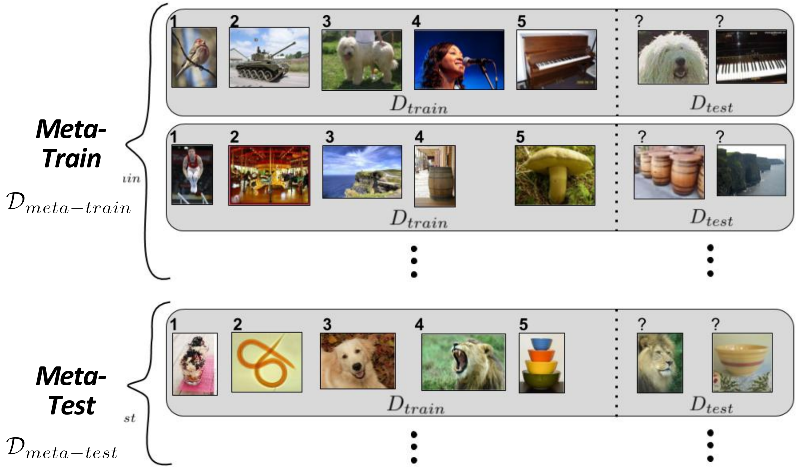




- Again, the meta-learner specifies subspace(dotted line) over initial weights(black) as chosen by task-specific learners(colored).



- Mini-ImageNet extracts 100 classes from ImageNet, and each class have 600 instances.
- MT-NET shows outperforming results over baselines.



Models	5-way 1-shot acc. (%)
Matching Networks(Vinyals et al., 2016)[1]	43.56 ± 0.84
Prototypical Networks(Snell et al., 2017)[2]	46.61 ± 0.78
mAP-SSVM(Triantafillou et al., 2017)	50.32 ± 0.80
Fine-tune baseline[1]	28.86 ± 0.54
Nearest Neighbor baseline[1]	41.08 ± 0.70
meta-learner LSTM(Ravi & Larochelle, 2017)	43.44 ± 0.77
MAML(Finn et al., 2017)	48.70 ± 1.84
L-MAML(Grant et al., 2018)	49.40 ± 1.83
Meta-SGD(Li et al., 2017)	50.47 ± 1.87
T-net (ours)	50.86 ± 1.82
MT-net (ours)	**51.70 ± 1.84**

Table of Contents

Optimizers for Learning DNNs

- 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]

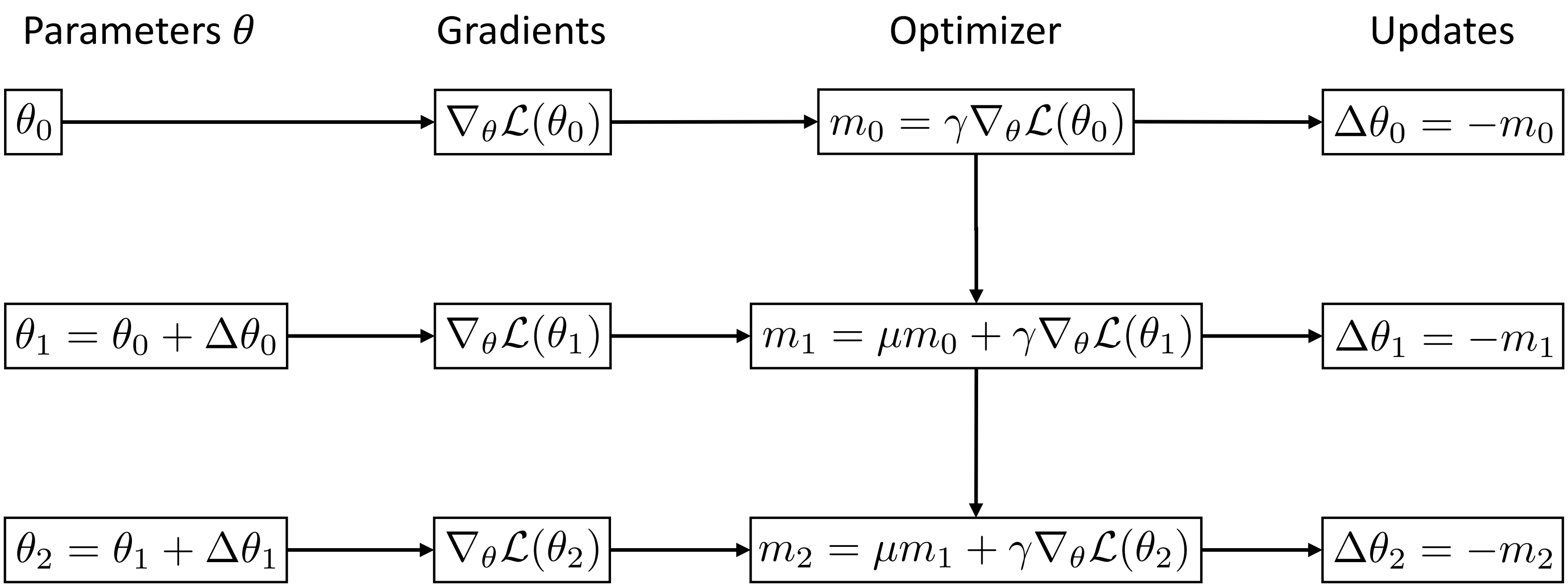
An Example of Optimizers: SGD with Momentum

- 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

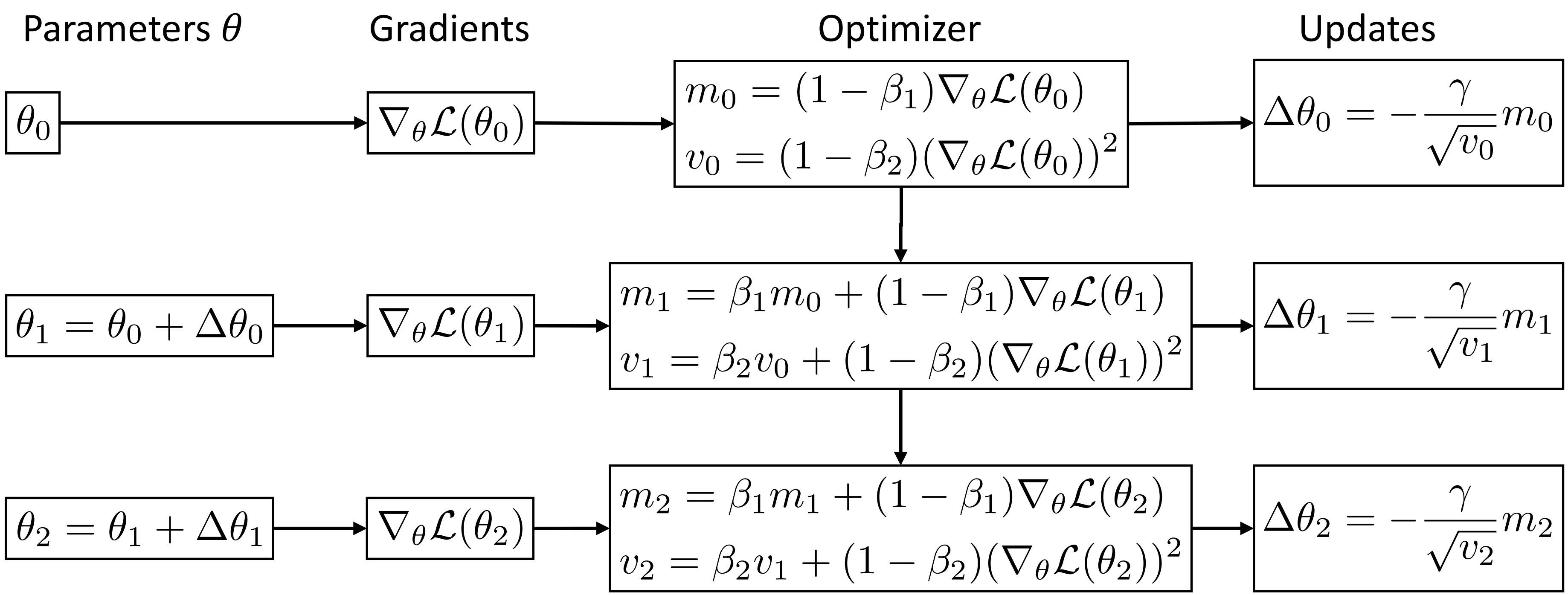


- Update rules of ADAM [Kingma and Ba, 2015]:

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

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

- Unroll the update steps



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**

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

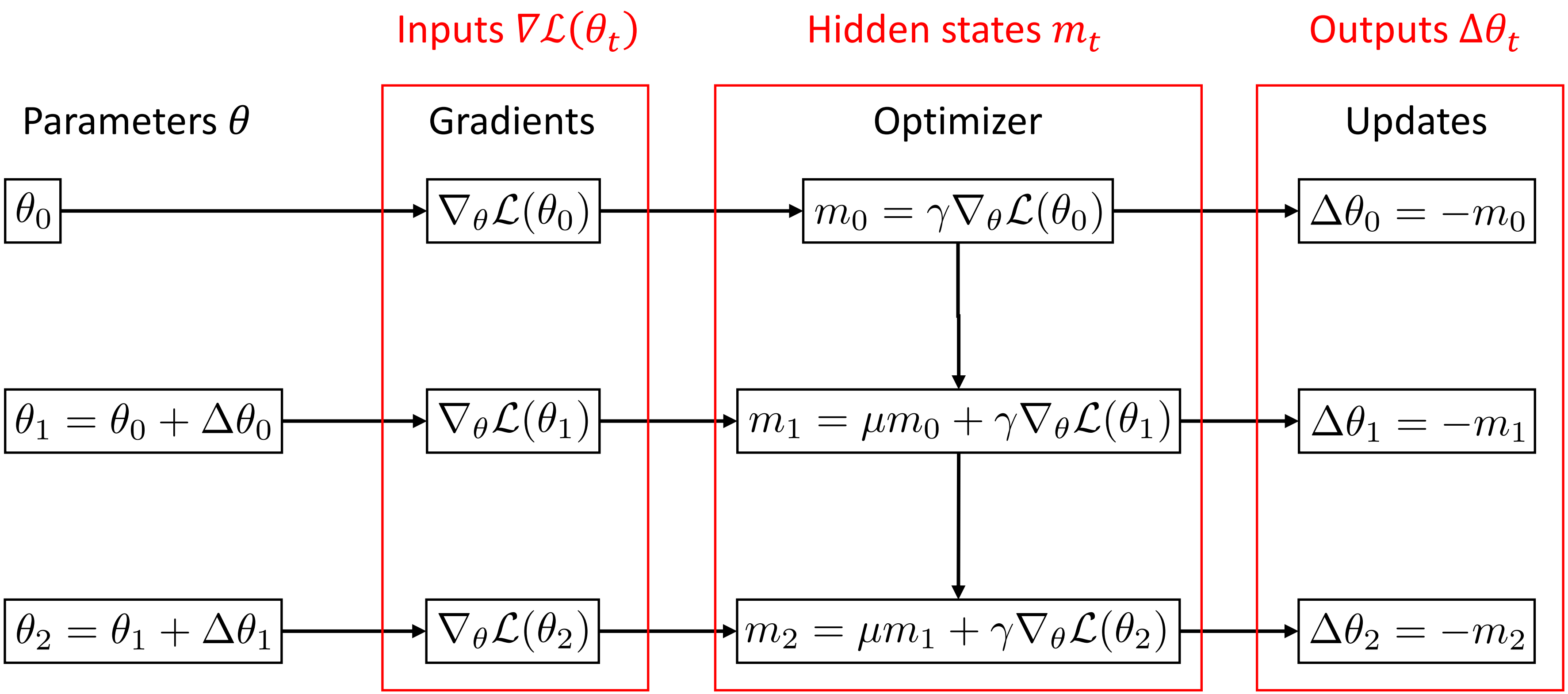
where $\theta_T(\phi)$ are the T-step updated parameters given the RNN optimizer ϕ

Recall: SGD with Momentum

- Update rules of SGD with momentum:

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

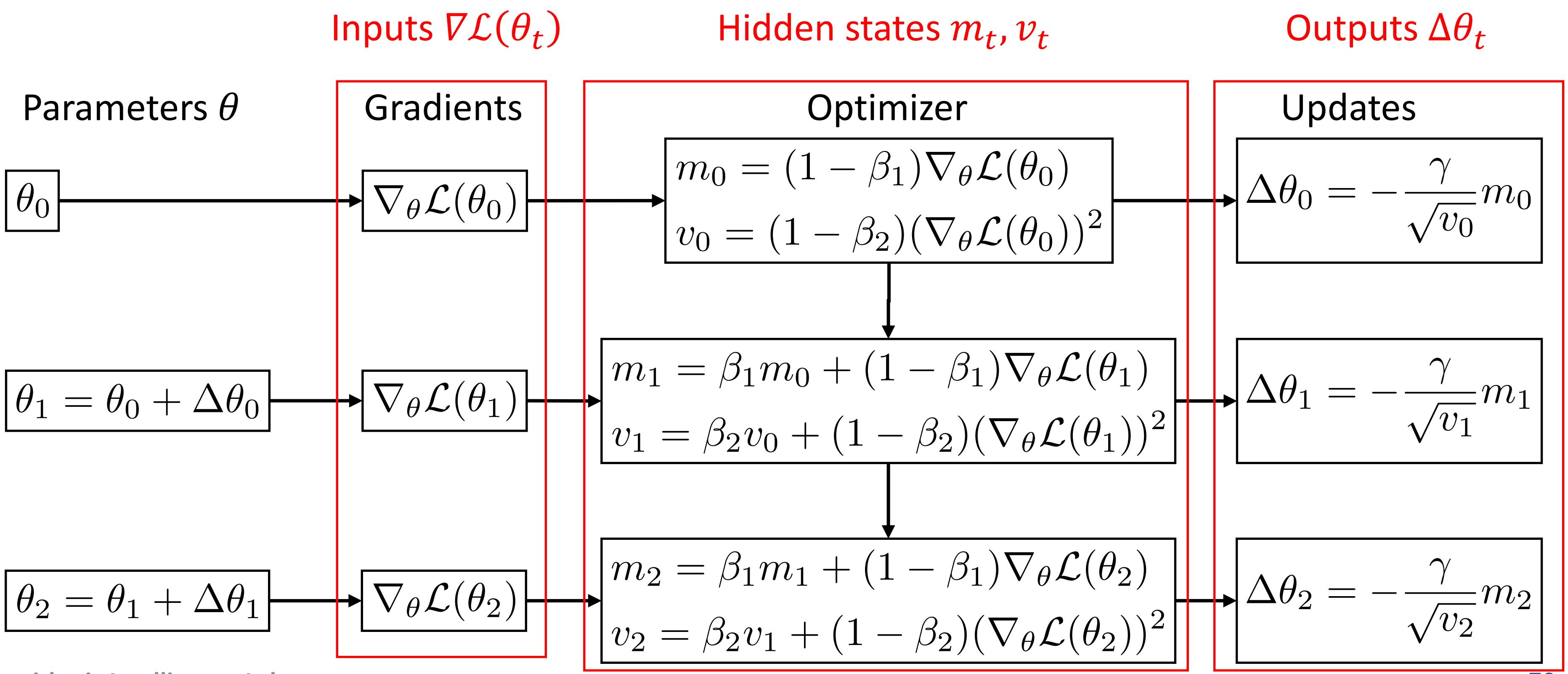
where γ is a learning rate and μ is a momentum



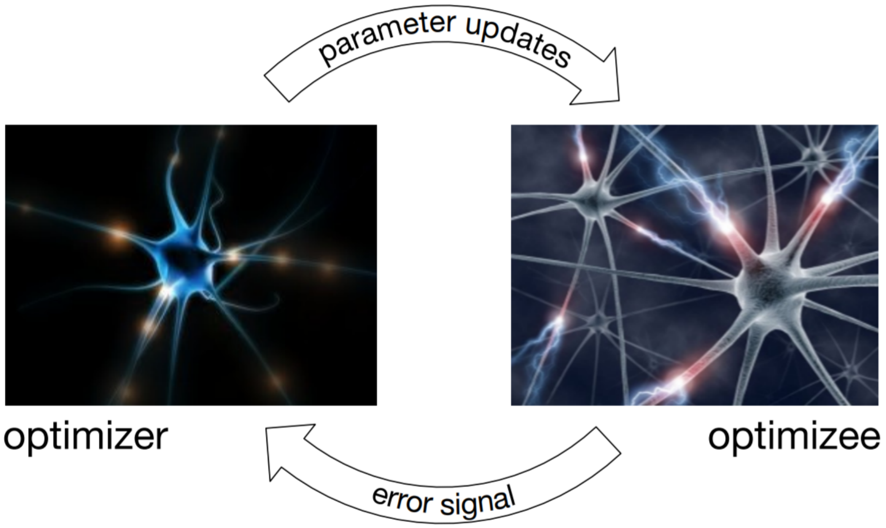
- Update rules of ADAM [Kingma and Ba, 2015]:

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

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



- [Andrychowicz et al. 16] proposes to learn the optimizer along with the learned model (optimizee).



$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$

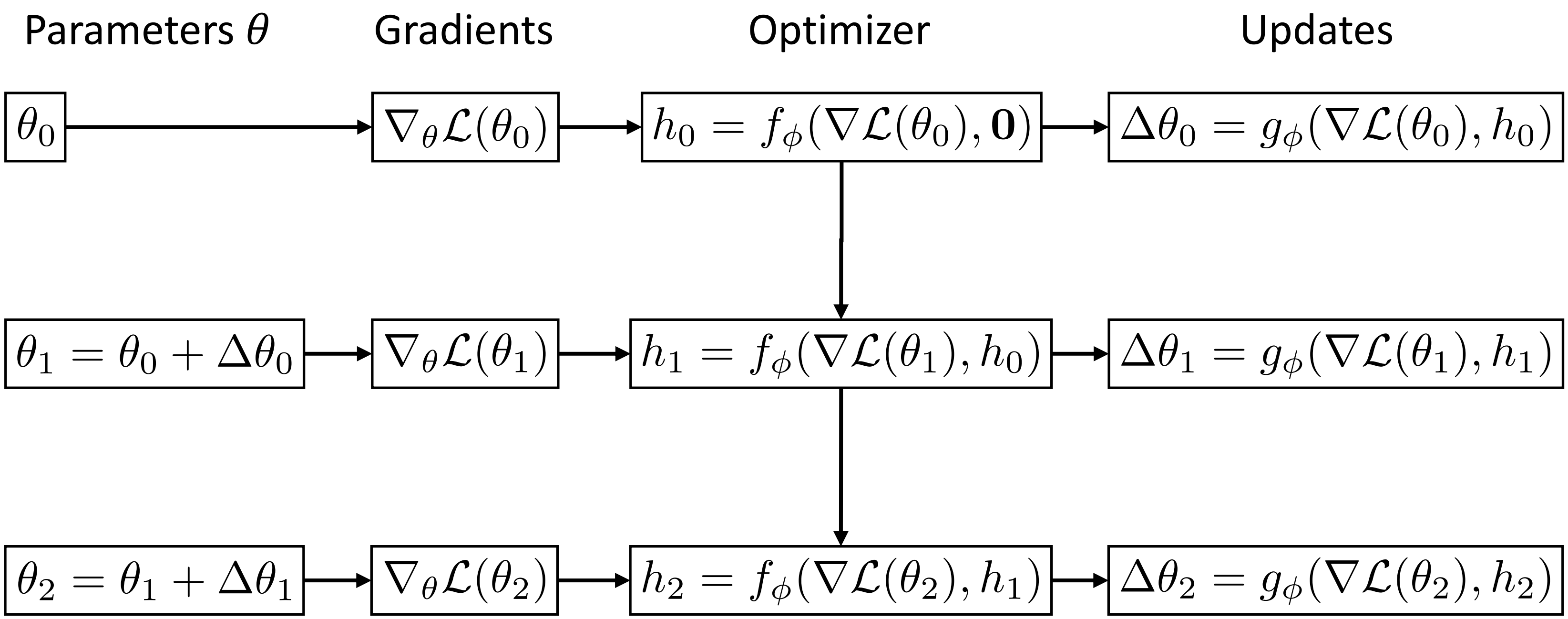
- The optimizer could be thought as a neural network parameterized with ϕ that receives the gradient at step t as an input, and generates the update $\Delta\theta$.

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

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

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

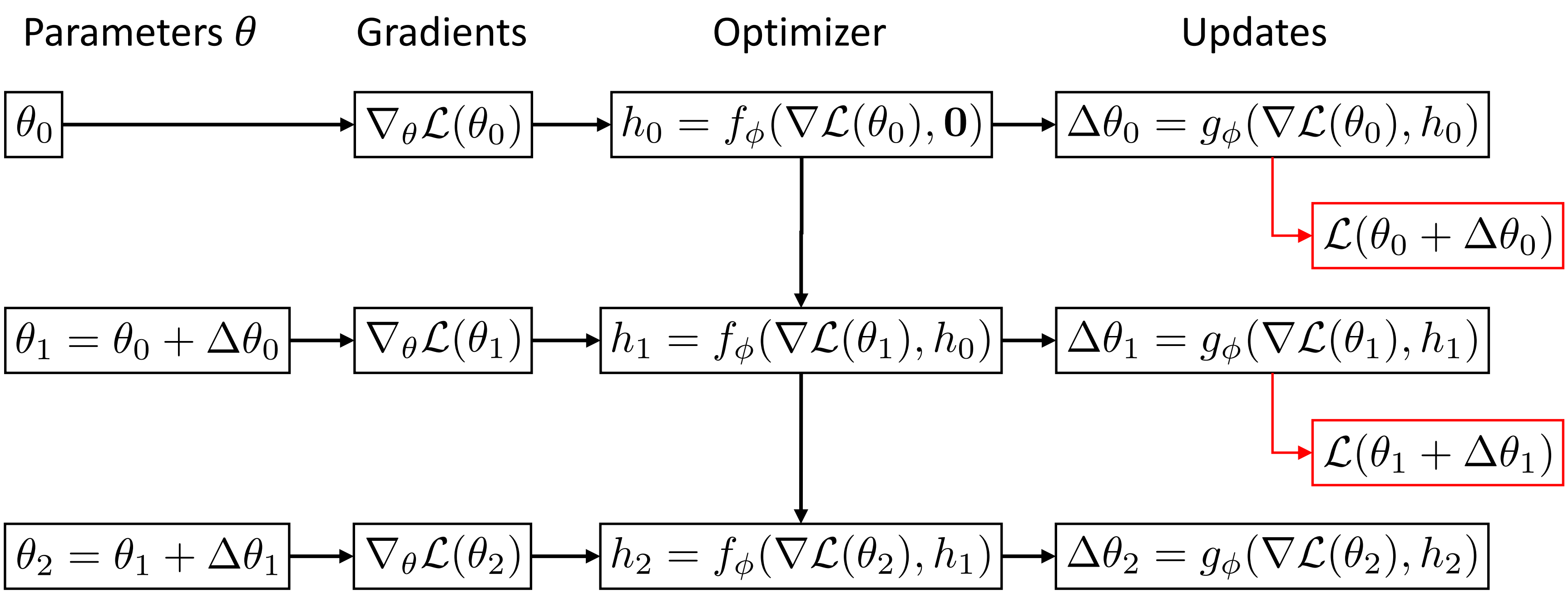
- **Inner-loop:** update the parameters θ via the optimizer for T times



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

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

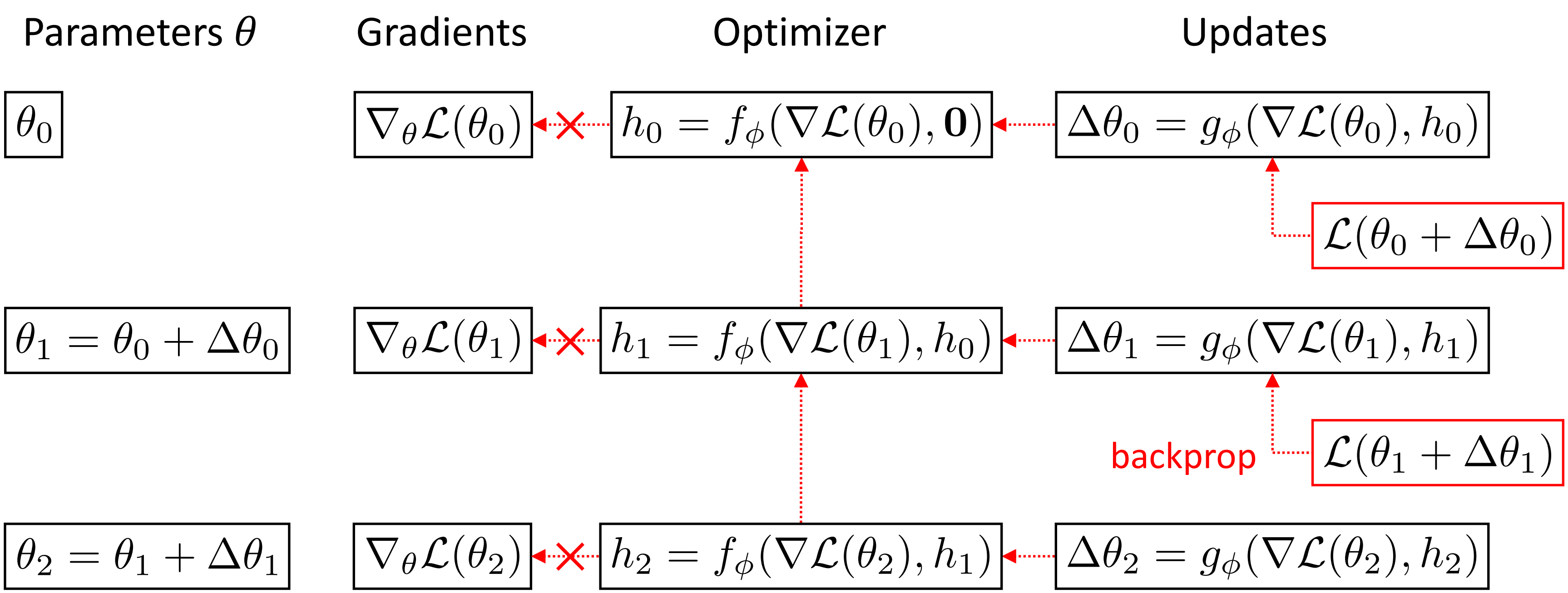
where w_t weights for each time-step



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

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

- **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)

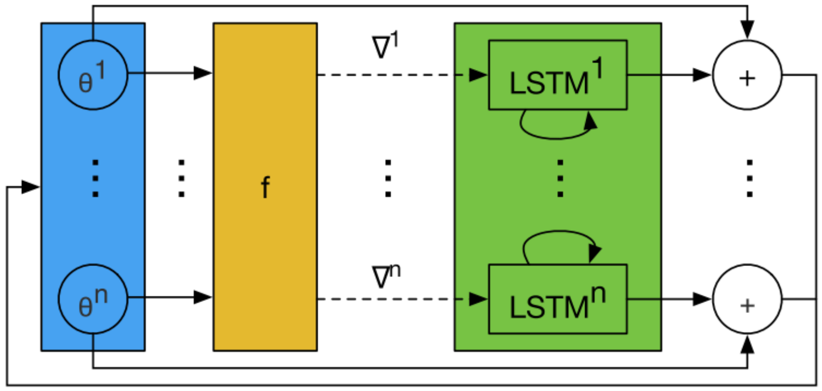


Architecture of RNN Optimizer

- 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



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

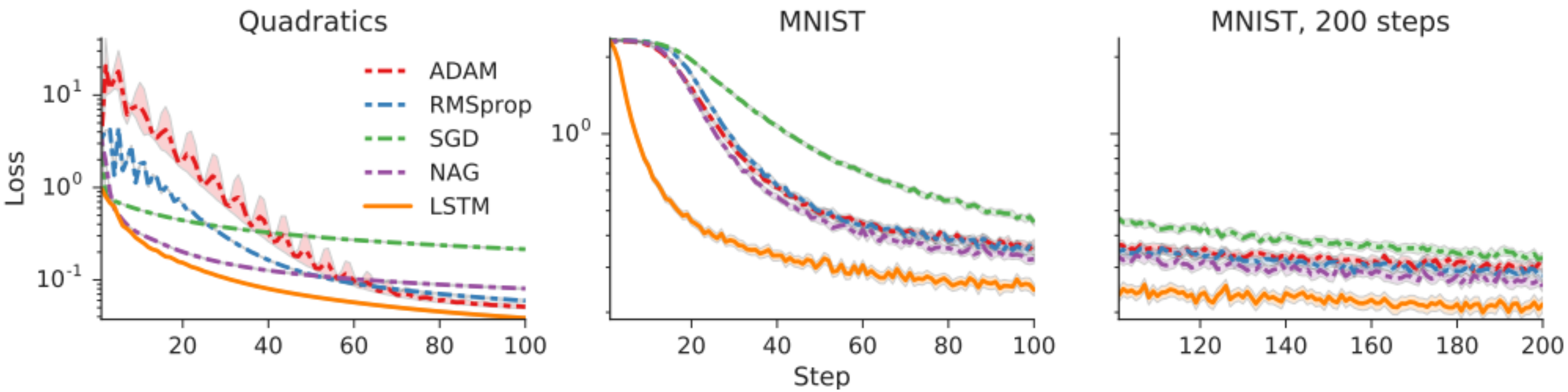
Effectiveness of a Learned Optimizer

- 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)

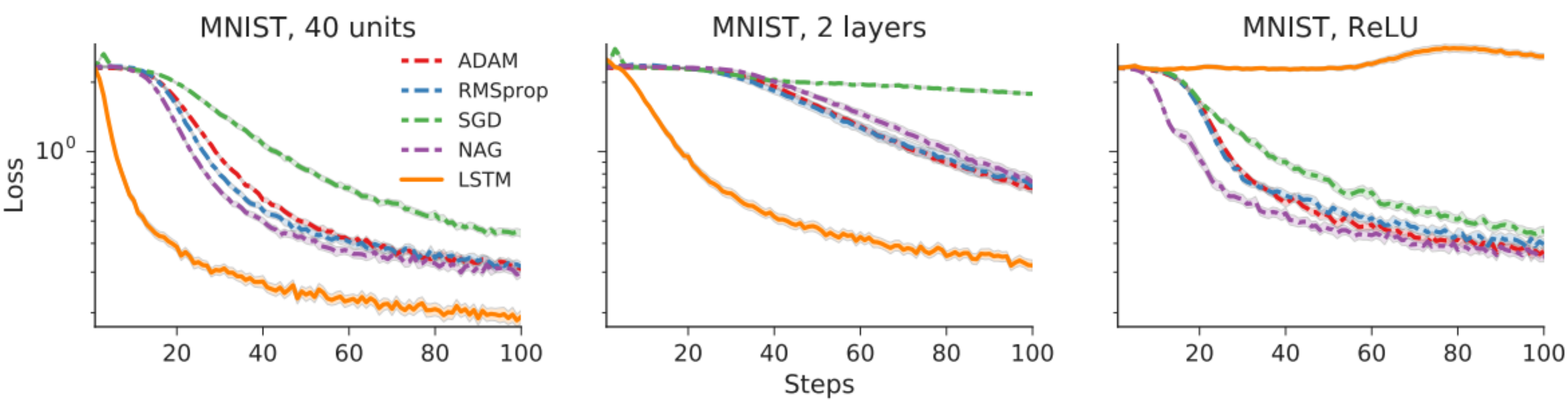
- Outperform baseline optimizers
  - Also perform well beyond the meta-trained steps (> 100 steps)



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

- **Generalization 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

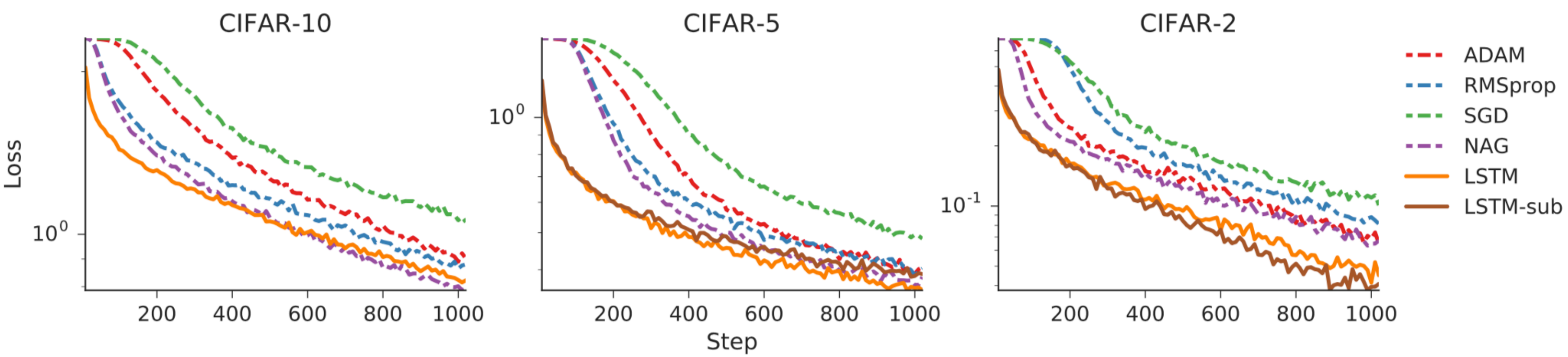


* source : Andrychowicz, et. al., Learning to learn by gradient descent by gradient descent, NIPS 2016

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

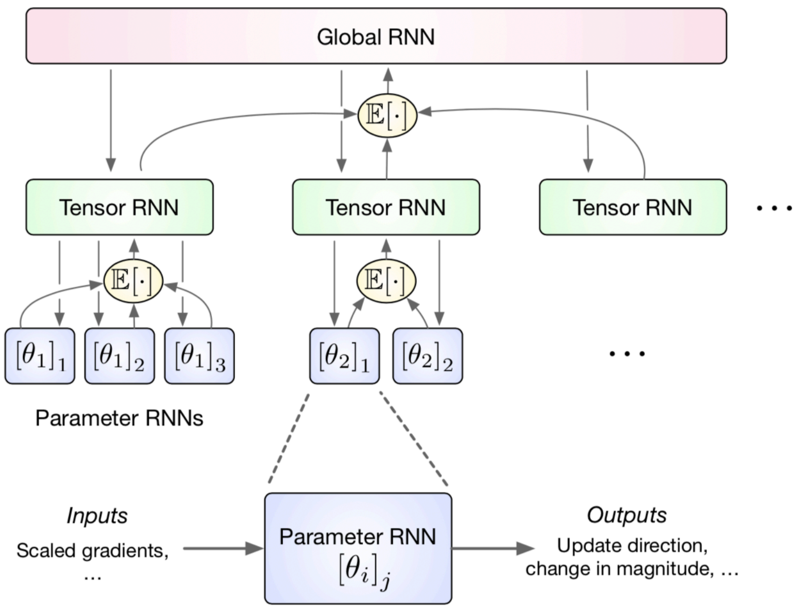


* source : Andrychowicz, et. al., Learning to learn by gradient descent by gradient descent, NIPS 2016

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

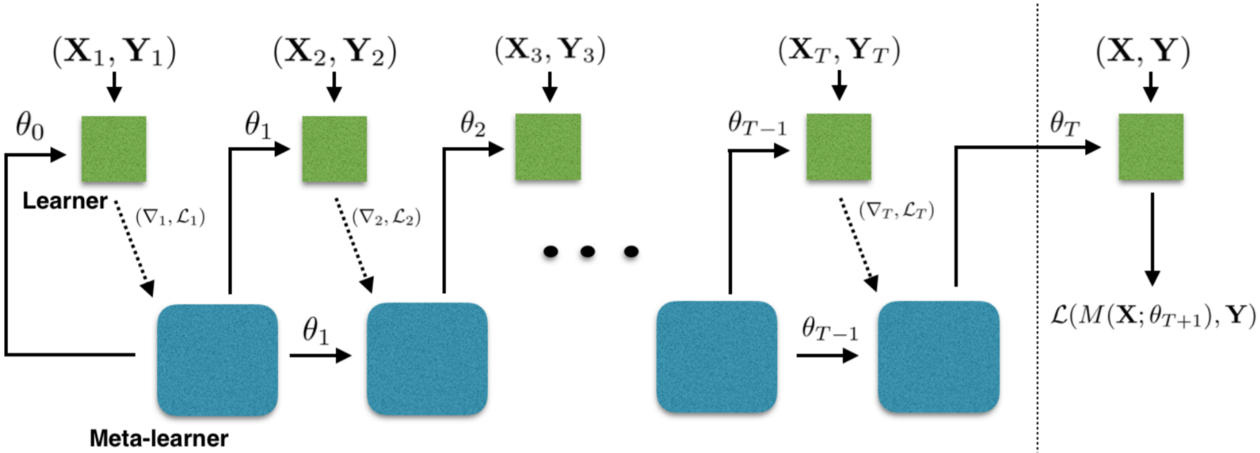
TensorRNN aggregates local information from a subset of ParameterRNNs

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

- [Ravi and Larochelle17] 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	$\mathbf{43.40 \pm 0.78\%}$	$51.09 \pm 0.71\%$
Matching Network FCE	$\mathbf{43.56 \pm 0.84\%}$	$55.31 \pm 0.73\%$
Meta-Learner LSTM (OURS)	$\mathbf{43.44 \pm 0.77\%}$	$\mathbf{60.60 \pm 0.71\%}$

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

[Ravi and Larochelle 17] Optimization as a Model for Few-shot Learning, ICLR 2017

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

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

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

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

References

[Lee, et. al., 2018] Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace, ICML 2018
https://arxiv.org/pdf/1902.04552.pdf

[A. Rusu, et. al., 2019] Meta-Learning with Latent Embedding Optimization, ICLR 2019
https://arxiv.org/pdf/1807.05960.pdf

[Allen, et. al., 2019] Infinite Mixture Prototypes for Few-Shot Learning, ICML 2019
https://arxiv.org/pdf/1902.04552.pdf

[Mishra et al. 18] A Simple Neural Attentive Meta-Learner, ICLR 2018
https://arxiv.org/pdf/1707.03141.pdf

[Garnelo et al. 18] Conditional Neural Process, ICML 2018
https://arxiv.org/pdf/1807.01613.pdf

[Lee et al. 19] Meta-Learning with Differentiable Convex Optimization, CVPR 2019
https://arxiv.org/pdf/1904.03758.pdf

[Rajeswaran et al. 19] Meta-Learning with Implicit Gradients, NeurIPS 2019
https://arxiv.org/pdf/1909.04630.pdf