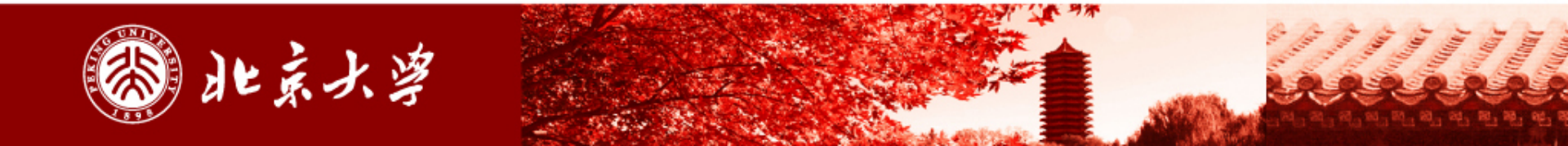
Meta Learning

— Learning to Learn

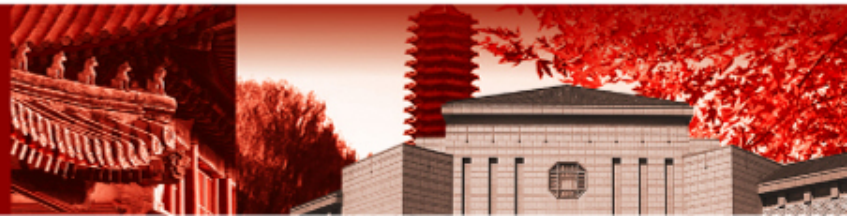
Wenpeng Hu
2019.4.9



Preface

Meta-learning, also known as "learning to learn", intends to design models that can learn new skills or adapt to new environments rapidly with a few training examples

A good machine learning model often requires training with a large number of samples. Humans, in contrast, learn new concepts and skills much faster and more efficiently. Kids who have seen cats and birds only a few times can quickly tell them apart. People who know how to ride a bike are likely to discover the way to ride a motorcycle fast with little or even no demonstration. Is it possible to design a machine learning model with similar properties — learning new concepts and skills fast with a few training examples? That's essentially what **meta-learning** aims to solve.



Preface

A Simple View

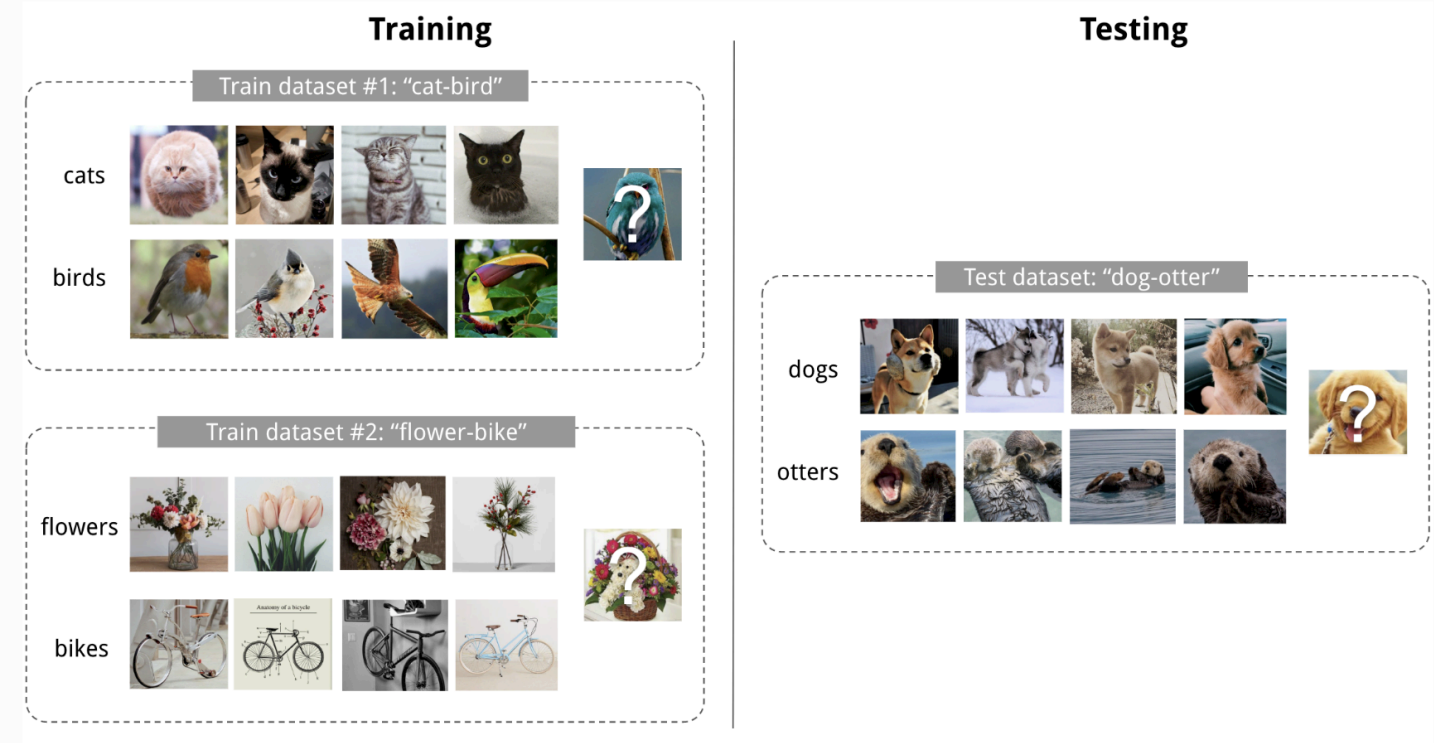
A good meta-learning model should be trained over a variety of learning tasks and optimized for the best performance on a distribution of tasks, including potentially unseen tasks. Each task is associated with a dataset $\mathcal{D}$, containing both feature vectors and true labels. The optimal model parameters are:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D})}[\mathcal{L}_\theta(\mathcal{D})]$$

It looks very similar to a normal learning task, but *one dataset* is considered as *one data sample*.

Preface

Few-shot classification is an instantiation of meta-learning in the field of supervised learning. The dataset $\mathcal{D}$ is often split into two parts, a support set S for learning and a prediction set B for training or testing, $\mathcal{D} = \langle S, B \rangle$. Often we consider a *K-shot N-class classification* task: the support set contains K labelled examples for each of N classes.



Preface

You may consider each pair of sampled dataset (S^L, B^L) as one data point. The model is trained such that it can generalize to other datasets. Symbols in red are added for meta-learning in addition to the supervised learning objective.

$$\theta = \arg\max_\theta E_{L\subset\mathcal{L}}[E_{S^L\subset\mathcal{D},B^L\subset\mathcal{D}}[\sum_{(x,y)\in B^L} P_\theta(x, y, S^L)]]$$

Learner and Meta-Learner

Another popular view of meta-learning decomposes the model update into two stages:

- A classifier f_θ is the "learner" model, trained for operating a given task;
- In the meantime, a optimizer g_ϕ learns how to update the learner model's parameters via the support set S, $\theta' = g_\phi(\theta, S)$.

Then in final optimization step, we need to update both θ and ϕ to maximize:

$$\mathbb{E}_{L\subset\mathcal{L}}[\mathbb{E}_{S^L\subset\mathcal{D},B^L\subset\mathcal{D}}[\sum_{(\mathbf{x},y)\in B^L} P_{g_\phi(\theta,S^L)}(y|\mathbf{x})]]$$

Outlines

There are three common approaches to meta-learning:

	Model-based	Metric-based	Optimization-based
Key idea	RNN; memory	Metric learning	Gradient descent
How $P_\theta(y\|\mathbf{x})$ is modeled?	$f_\theta(\mathbf{x}, S)$	$\sum_{(\mathbf{x}_i, y_i)\in S} k_\theta(\mathbf{x}, \mathbf{x}_i) y_i$ (*)	$P_{g_\phi(\theta, S^L)}(y\|\mathbf{x})$

(*) k_θ is a kernel function measuring the similarity between $\mathbf{x}_i$ and $\mathbf{x}$.

Outlines

Metric-Based

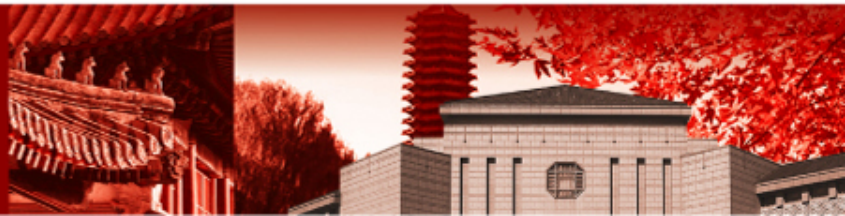
- Bromley, Jane, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature verification using a" siamese" time delay neural network." In *Advances in neural information processing systems*, pp. 737-744. 1994.
- Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." In *ICML deep learning workshop*, vol. 2. 2015.
- Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, and Daan Wierstra. "Matching networks for one shot learning." In *Advances in neural information processing systems*, pp. 3630-3638. 2016.
- Sung, Flood, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M. Hospedales. "Learning to compare: Relation network for few-shot learning." *In CVPR*, pp. 1199-1208. 2018.
- Snell, Jake, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning." In *Advances in Neural Information Processing Systems*, pp. 4077-4087. 2017.

Model-Based

- Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." *arXiv preprint arXiv:1410.5401* (2014).
- Santoro, Adam, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. "Meta-learning with memory-augmented neural networks." In *International conference on machine learning*, pp. 1842-1850. 2016.
- Munkhdalai, Tsendsuren, and Hong Yu. "Meta networks." In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 2554-2563. JMLR. org, 2017.

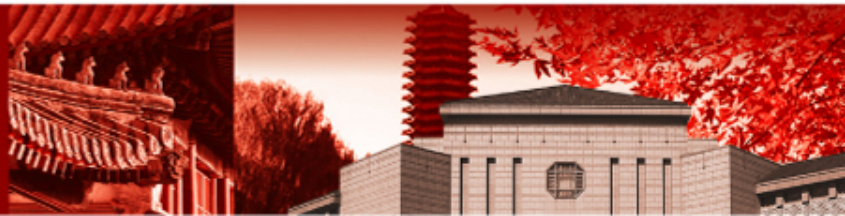
Optimization-Based

- Ravi, Sachin, and Hugo Larochelle. "Optimization as a model for few-shot learning." In *ICLR* (2017).
- Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.
- Nichol, Alex, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms." *arXiv preprint arXiv:1803.02999* (2018).



Chapter 1 Metric-Based

- **Bromley, Jane, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. "Signature verification using a" siamese" time delay neural network." In *Advances in neural information processing systems*, pp. 737-744. 1994.**
- **Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese neural networks for one-shot image recognition." In *ICML deep learning workshop*, vol. 2. 2015.**
- **Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, and Daan Wierstra. "Matching networks for one shot learning." In *Advances in neural information processing systems*, pp. 3630-3638. 2016.**
- **Sung, Flood, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M. Hospedales. "Learning to compare: Relation network for few-shot learning." *In CVPR*, pp. 1199-1208. 2018.**
- **Snell, Jake, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning." In *Advances in Neural Information Processing Systems*, pp. 4077-4087. 2017.**



Convolutional Siamese Neural Network *NIPS(2014)*

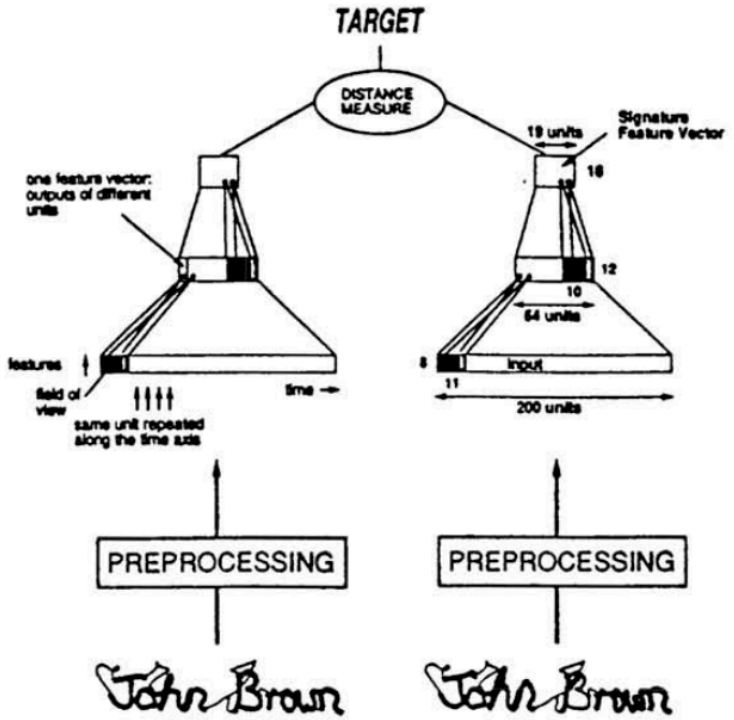


Figure 1: Architecture 1 consists of two identical time delay neural networks. Each network has an input of 8 by 200 units, first layer of 12 by 64 units with receptive fields for each unit being 8 by 11 and a second layer of 16 by 19 units with receptive fields 12 by 10.

Siamese Neural Networks for One-shot Image Recognition *ICML(2015)*

This paper proposed a method to use the siamese neural network to do one-shot image classification. First, the siamese network is trained for a verification task for telling whether two input images are in the same class. It outputs the probability of two images belonging to the same class. Then, during test time, the siamese network processes all the image pairs between a test image and every image in the support set. The final prediction is the class of the support image with the highest probability.

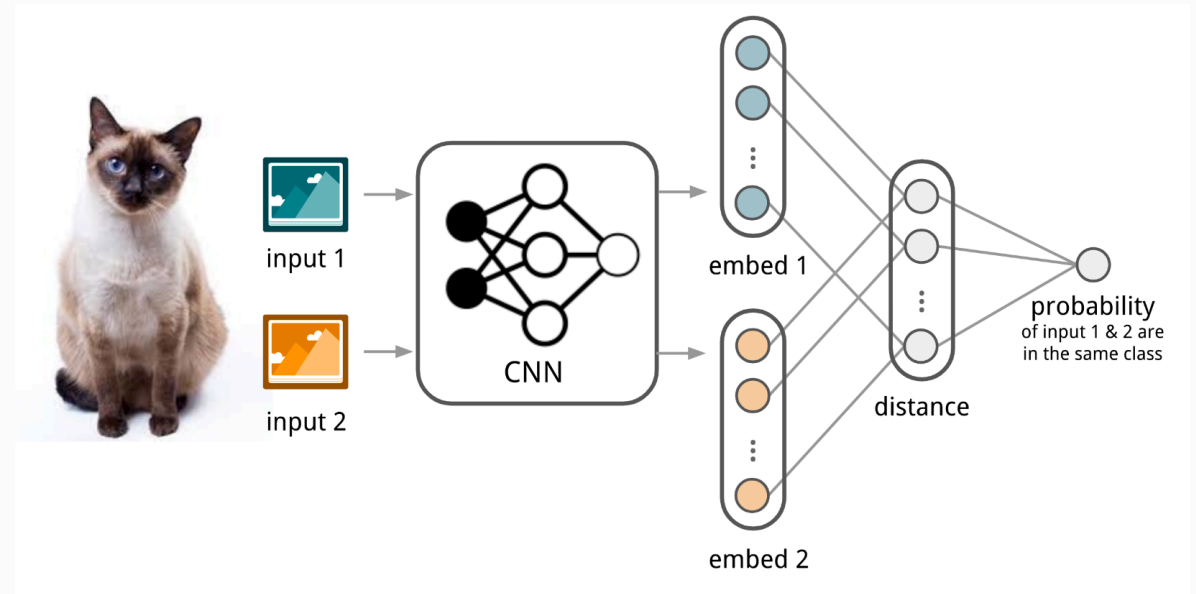


Fig. 2. The architecture of convolutional siamese neural network for few-show image classification.

Matching Networks for One Shot Learning *NIPS(2016)*

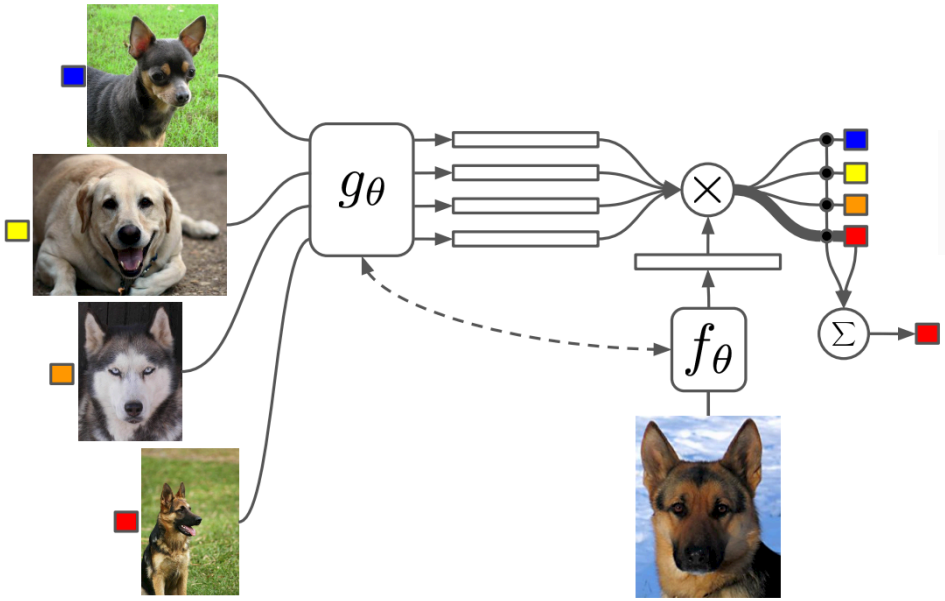


Figure 1: Matching Networks architecture

$$c_S(\mathbf{x}) = P(y|\mathbf{x}, S) = \sum_{i=1}^{k} a(\mathbf{x}, \mathbf{x}_i) y_i, \text{ where } S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{k}$$

$$a(\mathbf{x}, \mathbf{x}_i) = \frac{\exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_i))}{\sum_{j=1}^{k} \exp(\text{cosine}(f(\mathbf{x}), g(\mathbf{x}_j))}$$

Matching Networks for One Shot Learning *NIPS(2016)*

- $g_\theta(\mathbf{x}_i, S)$ uses a bidirectional LSTM to encode $\mathbf{x}_i$ in the context of the entire support set S.
- $f_\theta(\mathbf{x}, S)$ encodes the test sample $\mathbf{x}$ visa an LSTM with read attention over the support set S.
  1. First the test sample goes through a simple neural network, such as a CNN, to extract basic features, $f'(\mathbf{x})$.
  2. Then an LSTM is trained with a read attention vector over the support set as part of the hidden state:

$$\hat{\mathbf{h}}_t, \mathbf{c}_t = \text{LSTM}(f'(\mathbf{x}), [\mathbf{h}_{t-1}, \mathbf{r}_{t-1}], \mathbf{c}_{t-1})$$
$$\mathbf{h}_t = \hat{\mathbf{h}}_t + f'(\mathbf{x})$$
$$\mathbf{r}_{t-1} = \sum_{i=1}^{k} a(\mathbf{h}_{t-1}, g(\mathbf{x}_i)) g(\mathbf{x}_i)$$
$$a(\mathbf{h}_{t-1}, g(\mathbf{x}_i)) = \text{softmax}(\mathbf{h}_{t-1}^\top g(\mathbf{x}_i)) = \frac{\exp(\mathbf{h}_{t-1}^\top g(\mathbf{x}_i))}{\sum_{j=1}^{k} \exp(\mathbf{h}_{t-1}^\top g(\mathbf{x}_j))}$$

  3. Eventually $f(\mathbf{x}, S) = \mathbf{h}_K$ if we do K steps of “read”.

Learning to Compare: Relation Network for Few-Shot Learning *CVPR(2018)*

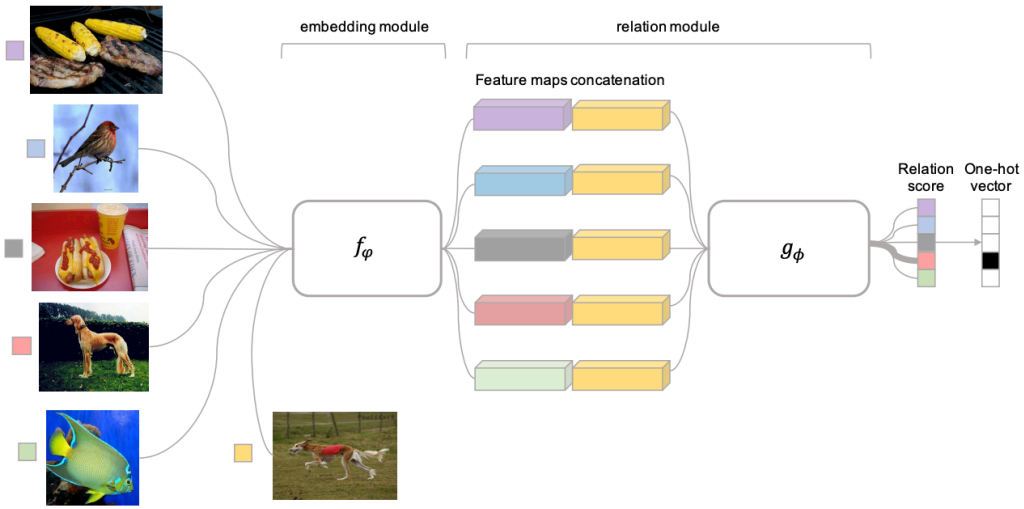
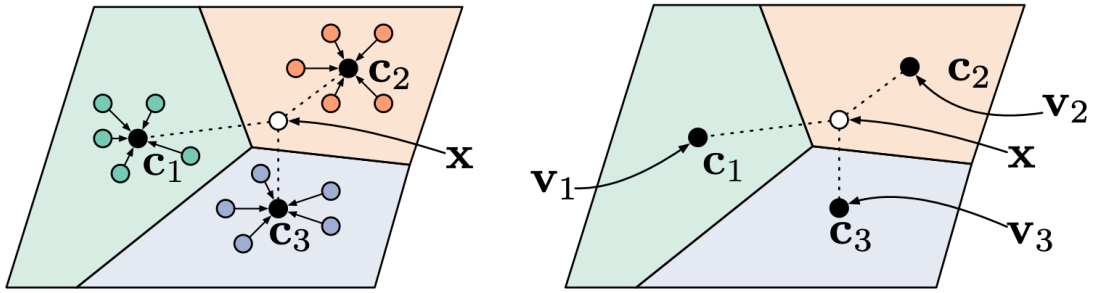


Figure 1: Relation Network architecture for a 5-way 1-shot problem with one query example.

1. The relationship is not captured by a simple L1 distance in the feature space, but predicted by a CNN classifier g_ϕ. The relation score between a pair of inputs, $\mathbf{x}_i$ and $\mathbf{x}_j$, is $r_{ij} = g_\phi([\mathbf{x}_i, \mathbf{x}_j])$ where $[.\,,.]$ is concatenation.
2. The objective function is MSE loss instead of cross-entropy, because conceptually RN focuses more on predicting relation scores which is more like regression, rather than binary classification, $\mathcal{L}(B) = \sum_{(\mathbf{x}_i,\mathbf{x}_j,y_i,y_j)\in B}(r_{ij} - \mathbf{1}_{y_i=y_j})^2$.

Prototypical Networks for Few-shot Learning *NIPS(2017)*



(a) Few-shot (b) Zero-shot

Figure 1: Prototypical Networks in the few-shot and zero-shot scenarios. **Left**: Few-shot prototypes $\mathbf{c}_k$ are computed as the mean of embedded support examples for each class. **Right**: Zero-shot prototypes $\mathbf{c}_k$ are produced by embedding class meta-data $\mathbf{v}_k$. In either case, embedded query points are classified via a softmax over distances to class prototypes: $p_{\phi}(y = k|\mathbf{x}) \propto \exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_k))$.

Prototypical Networks compute an M-dimensional representation $\mathbf{c}_k \in \mathbb{R}^M$, or *prototype*, of each class through an embedding function $f_{\phi} : \mathbb{R}^D \rightarrow \mathbb{R}^M$ with learnable parameters ϕ. Each prototype is the mean vector of the embedded support points belonging to its class:

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

Given a distance function $d : \mathbb{R}^M \times \mathbb{R}^M \rightarrow [0, +\infty)$, Prototypical Networks produce a distribution over classes for a query point $\mathbf{x}$ based on a softmax over distances to the prototypes in the embedding space:

$$p_{\phi}(y = k \,|\, \mathbf{x}) = \frac{\exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_{k'}))} \tag{2}$$

Chapter 2 Model-Based

- **Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." *arXiv preprint arXiv:1410.5401* (2014).**

- **Santoro, Adam, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. "Meta-learning with memory-augmented neural networks." In *International conference on machine learning*, pp. 1842-1850. 2016.**

- **Munkhdalai, Tsendsuren, and Hong Yu. "Meta networks." In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 2554-2563. JMLR. org, 2017.**

Neural turing machines. arXiv*(2014)*

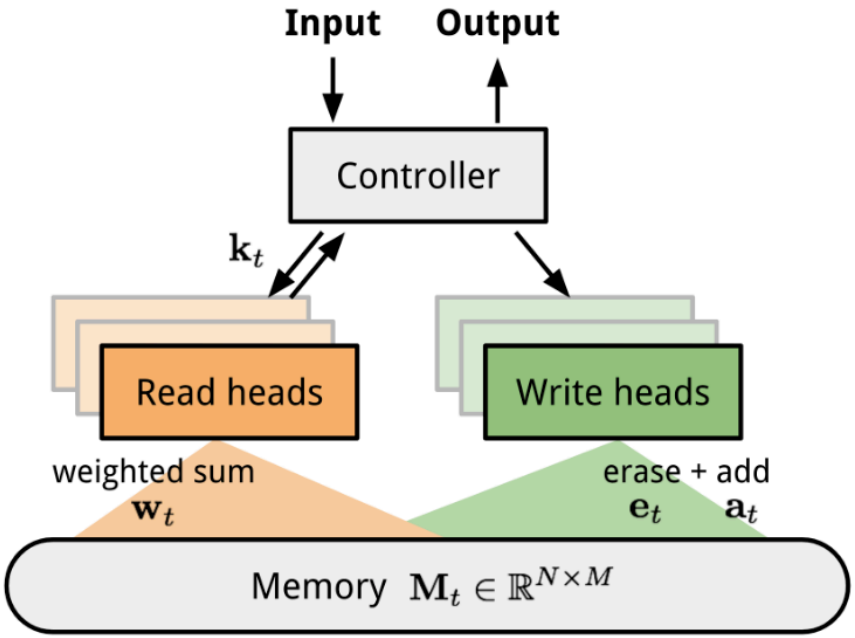


Fig. 6. The architecture of Neural Turing Machine (NTM). The memory at time t, $\mathbf{M}_t$ is a matrix of size $N \times M$, containing N vector rows and each has M dimensions.

Meta-Learning with Memory-Augmented Neural Networks. *ICML(2016)*

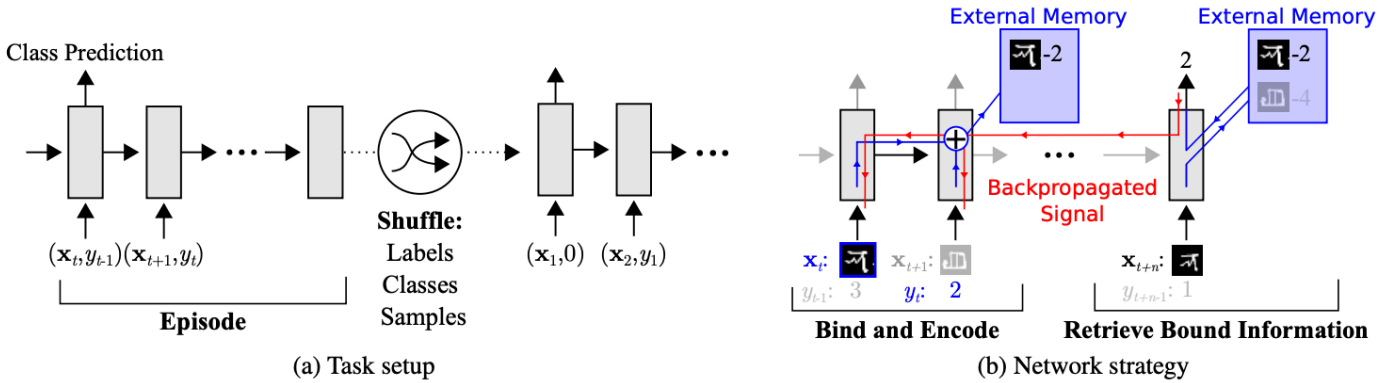


Figure 1. Task structure. (a) Omniglot images (or x-values for regression), $\mathbf{x}_t$, are presented with time-offset labels (or function values), y_{t-1}, to prevent the network from simply mapping the class labels to the output. From episode to episode, the classes to be presented in the episode, their associated labels, and the specific samples are all shuffled. (b) A successful strategy would involve the use of an external memory to store bound sample representation-class label information, which can then be retrieved at a later point for successful classification when a sample from an already-seen class is presented. Specifically, sample data x_t from a particular time step should be bound to the appropriate class label y_t, which is presented in the subsequent time step. Later, when a sample from this same class is seen, it should retrieve this bound information from the external memory to make a prediction. Backpropagated error signals from this prediction step will then shape the weight updates from the earlier steps in order to promote this binding strategy.

Meta Networks. *ICML(2017)*

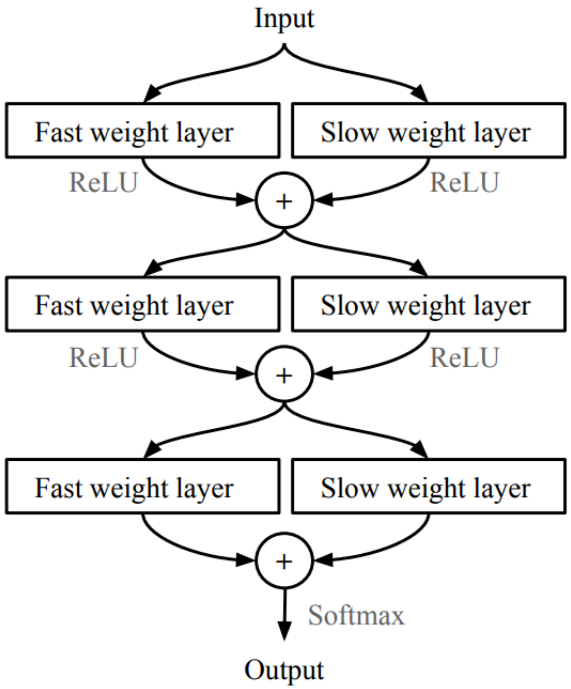


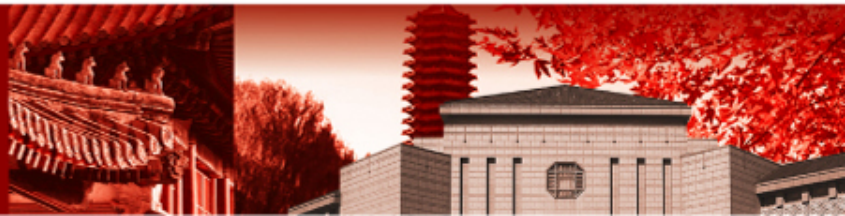
Figure 2. A layer augmented MLP

Table 1. One-shot accuracy on Omniglot previous split

Model	**5-way**	**10-way**	**15-way**	**20-way**
Pixel kNN (Kaiser et al., 2017)	41.7	-	-	26.7
Siamese Net (Koch, 2015)	97.3	-	-	88.1
MANN (Santoro et al., 2016)	82.8	-	-	-
Matching Nets (Vinyals et al., 2016)	98.1	-	-	93.8
Neural Statistician (Edwards & Storkey, 2017)	98.1	-	-	93.2
Siamese Net with Memory (Kaiser et al., 2017)	98.4	-	-	95.0
MetaNet-	98.4	98.32	96.68	96.13
MetaNet	**98.95**	**98.67**	**97.11**	**97.0**
MetaNet+	98.45	97.05	96.48	95.08

Chapter 3 Optimization-Based

- **Ravi, Sachin, and Hugo Larochelle. "Optimization as a model for few-shot learning." In *ICLR* (2017).**
- **Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.**
- **Nichol, Alex, Joshua Achiam, and John Schulman. "On first-order meta-learning algorithms." *arXiv preprint arXiv:1803.02999* (2018).**



OPTIMIZATION AS A MODEL FOR FEW-SHOT LEARNING *ICLR(2017)*

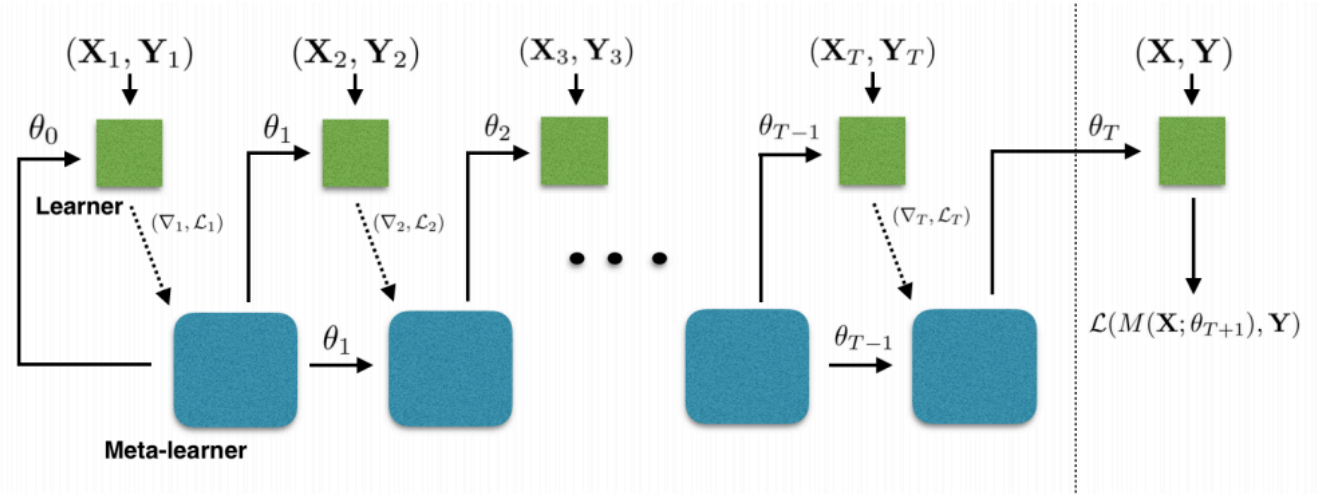


Figure 2: Computational graph for the forward pass of the meta-learner. The dashed line divides examples from the training set D_{train} and test set D_{test}. Each $(\mathbf{X}_i, \mathbf{Y}_i)$ is the i^{th} batch from the training set whereas $(\mathbf{X}, \mathbf{Y})$ is all the elements from the test set. The dashed arrows indicate that we do not back-propagate through that step when training the meta-learner. We refer to the learner as M, where $M(\mathbf{X}; \theta)$ is the output of learner M using parameters θ for inputs $\mathbf{X}$. We also use ∇_t as a shorthand for $\nabla_{\theta_{t-1}} \mathcal{L}_t$.

OPTIMIZATION AS A MODEL FOR FEW-SHOT LEARNING *ICLR(2017)*

The update for the learner's parameters at time step t with a learning rate α_t is:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

It has the same form as the cell state update in LSTM, if we set forget gate $f_t = 1$, input gate $i_t = \alpha_t$, cell state $c_t = \theta_t$, and new cell state $\tilde{c}_t = -\nabla_{\theta_{t-1}} \mathcal{L}_t$:

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ &= \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \end{aligned}$$

While fixing $f_t = 1$ and $i_t = \alpha_t$ might not be the optimal, both of them can be learnable and adaptable to different datasets.

$$\begin{aligned} f_t &= \sigma(\mathbf{W}_f \cdot [\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}] + \mathbf{b}_f) && \text{; how much to forget the old value of parameters.} \\ i_t &= \sigma(\mathbf{W}_i \cdot [\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}] + \mathbf{b}_i) && \text{; corresponding to the learning rate at time step t.} \\ \tilde{\theta}_t &= -\nabla_{\theta_{t-1}} \mathcal{L}_t \\ \theta_t &= f_t \odot \theta_{t-1} + i_t \odot \tilde{\theta}_t \end{aligned}$$

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks *ICML(2017)*

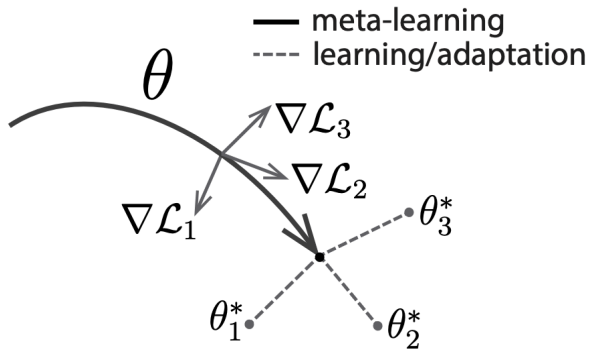


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

Algorithm 1 Model-Agnostic Meta-Learning

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

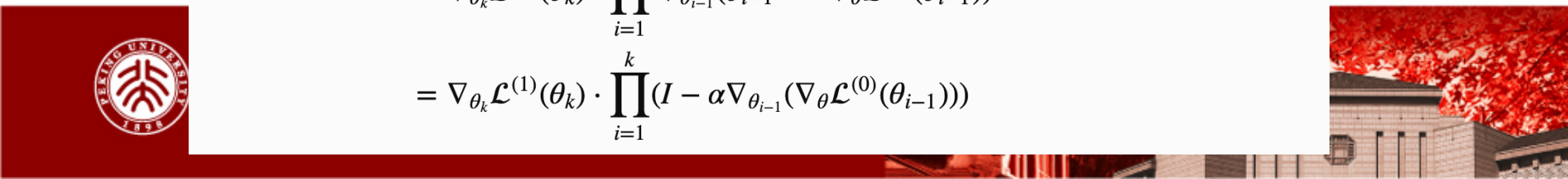
On first-order meta-learning algorithms *arXiv(2018)*

Let's consider the case of performing k inner gradient steps, $k \geq 1$. Starting with the initial model parameter θ_{meta}:

$$
\begin{aligned}
\theta_0 &= \theta_{\text{meta}} \\
\theta_1 &= \theta_0 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_0) \\
\theta_2 &= \theta_1 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_1) \\
&\dots \\
\theta_k &= \theta_{k-1} - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_{k-1})
\end{aligned}
$$

Then in the outer loop, we sample a new data batch for updating the meta-objective.

$$
\begin{aligned}
\theta_{\text{meta}} &\leftarrow \theta_{\text{meta}} - \beta g_{\text{MAML}} && \text{; update for meta-objective} \\
\text{where } g_{\text{MAML}} &= \nabla_\theta \mathcal{L}^{(1)}(\theta_k) \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot (\nabla_{\theta_{k-1}} \theta_k) \dots (\nabla_{\theta_0} \theta_1) \cdot (\nabla_\theta \theta_0) && \text{; following the chain rule} \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} \nabla_{\theta_{i-1}} \theta_i \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} \nabla_{\theta_{i-1}} (\theta_{i-1} - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_{i-1})) \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} (I - \alpha \nabla_{\theta_{i-1}} (\nabla_\theta \mathcal{L}^{(0)}(\theta_{i-1})))
\end{aligned}
$$



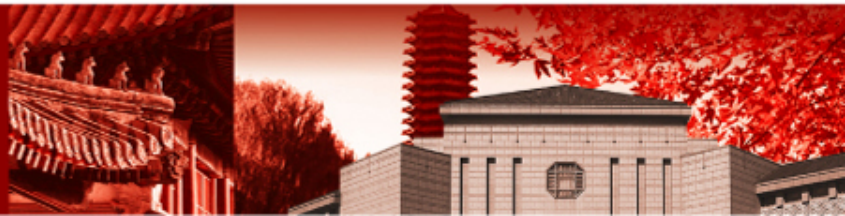
On first-order meta-learning algorithms *arXiv(2018)*

The MAML gradient is:

$$g_{\text{MAML}} = \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} (I - \alpha \nabla_{\theta_{i-1}} (\nabla_{\theta} \mathcal{L}^{(0)}(\theta_{i-1})))$$

The First-Order MAML ignores the second derivative part in red. It is simplified as follows, equivalent to the derivative of the last inner gradient update result.

$$g_{\text{FOMAML}} = \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k)$$



On first-order meta-learning algorithms *arXiv(2018)*

The Reptile works by repeatedly:

- 1) sampling a task,
- 2) training on it by multiple gradient descent steps,
- 3) and then moving the model weights towards the new parameters.

See the algorithm below: $\mathrm{SGD}(\mathcal{L}_{\tau_i}, \theta, k)$ performs stochastic gradient update for k steps on the loss $\mathcal{L}_{\tau_i}$ starting with initial parameter θ and returns the final parameter vector. The batch version samples multiple tasks instead of one within each iteration. The reptile gradient is defined as $(\theta - W)/\alpha$, where α is the stepsize used by the SGD operation.

Algorithm 2 Reptile, batched version

Initialize θ
for iteration $= 1, 2, \ldots$ **do**
  Sample tasks $\tau_1, \tau_2, \ldots, \tau_n$
  for $i = 1, 2, \ldots, n$ **do**
    Compute $W_i = \mathrm{SGD}(L_{\tau_i}, \theta, k)$
  end for
  Update $\theta \leftarrow \theta + \beta \frac{1}{n} \sum_{i=1}^{n} (W_i - \theta)$
end for

谢 谢!

北京大学