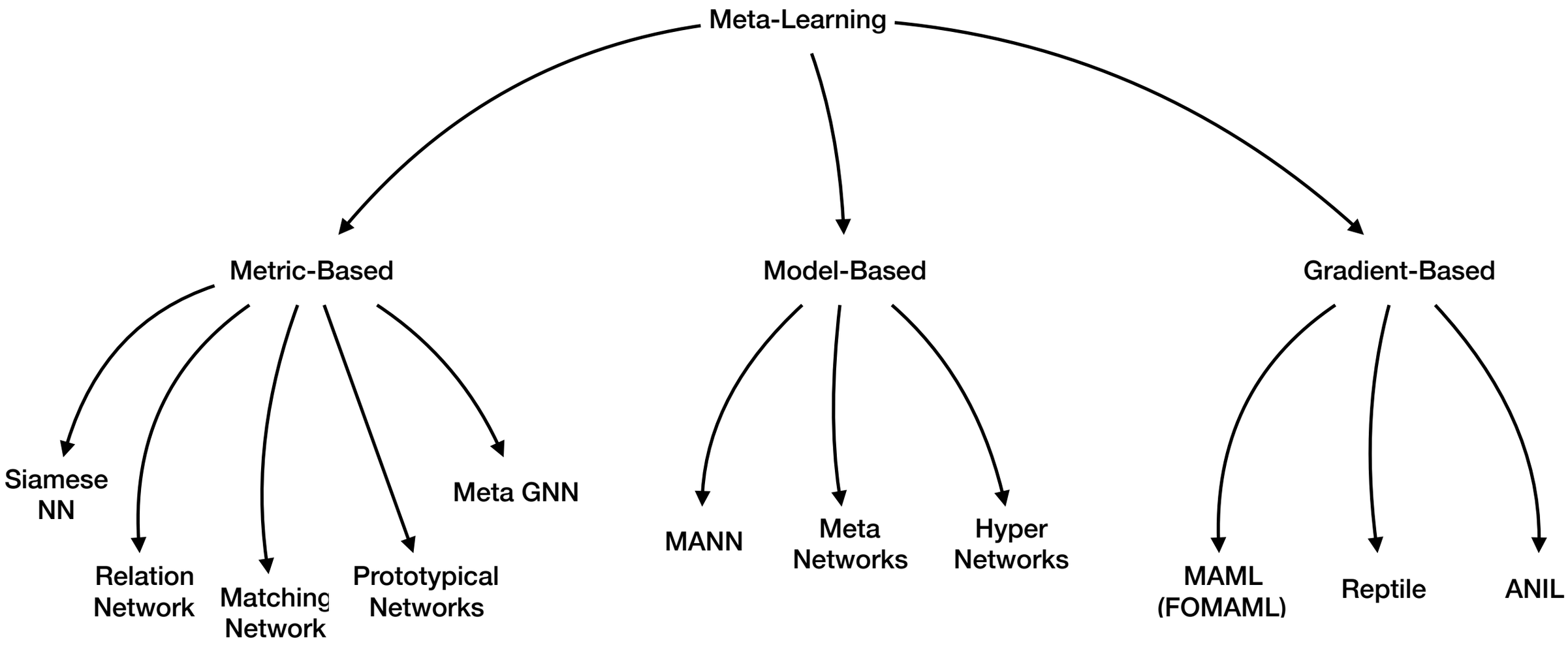
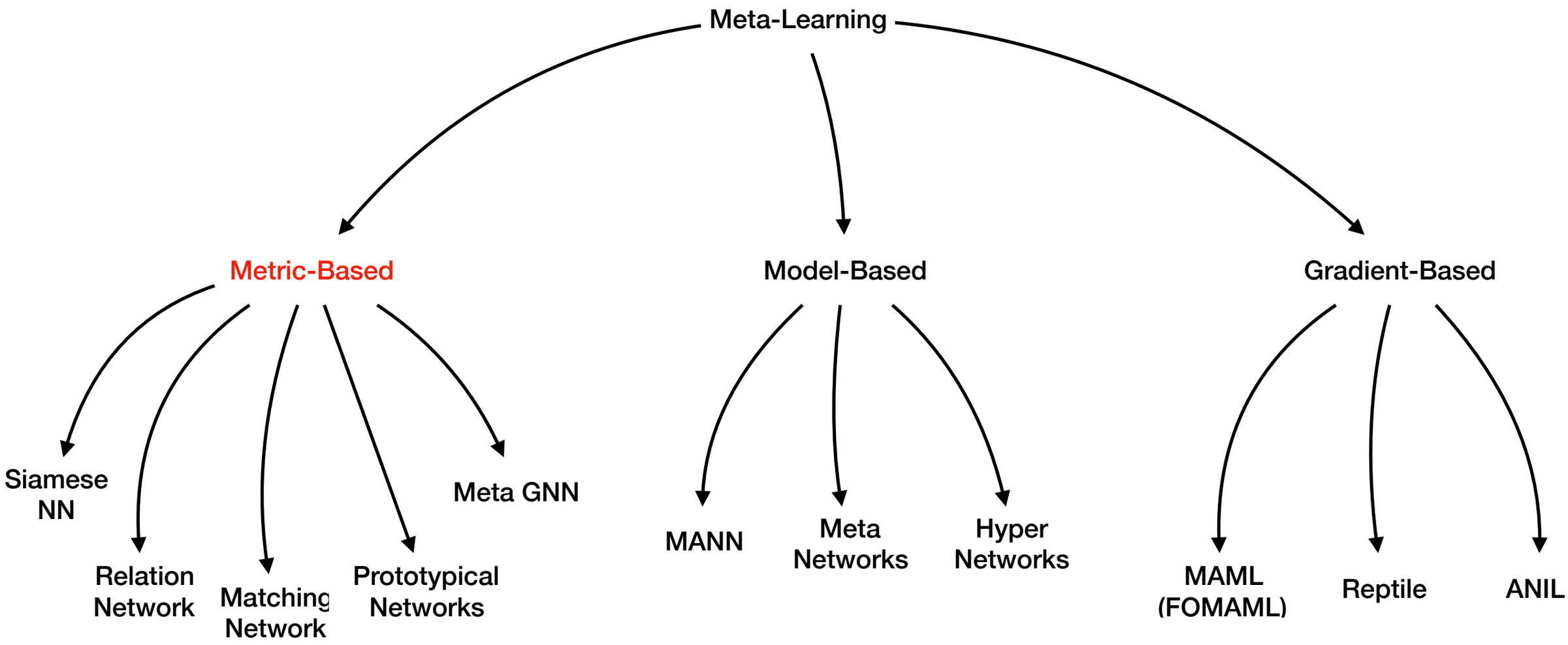
Meta Learning

Shengchao Liu

Background

- Meta Learning (AKA Learning to Learn)
  - A fast-learning algorithm: quickly adapted from the source tasks to the target tasks
- Key terminologies
  - Support Set & Query Set
  - C-Way K-Shot Learning: C classes and each with K samples
  - Pre-training & Fine-tuning


Meta-Learning
Metric-Based
Model-Based
Gradient-Based
Siamese NN
Relation Network
Matching Network
Prototypical Networks
Meta GNN
MANN
Meta Networks
Hyper Networks
MAML (FOMAML)
Reptile
ANIL


Meta-Learning
Metric-Based
Model-Based
Gradient-Based
Siamese NN
Relation Network
Matching Network
Prototypical Networks
Meta GNN
MANN
Meta Networks
Hyper Networks
MAML (FOMAML)
Reptile
ANIL

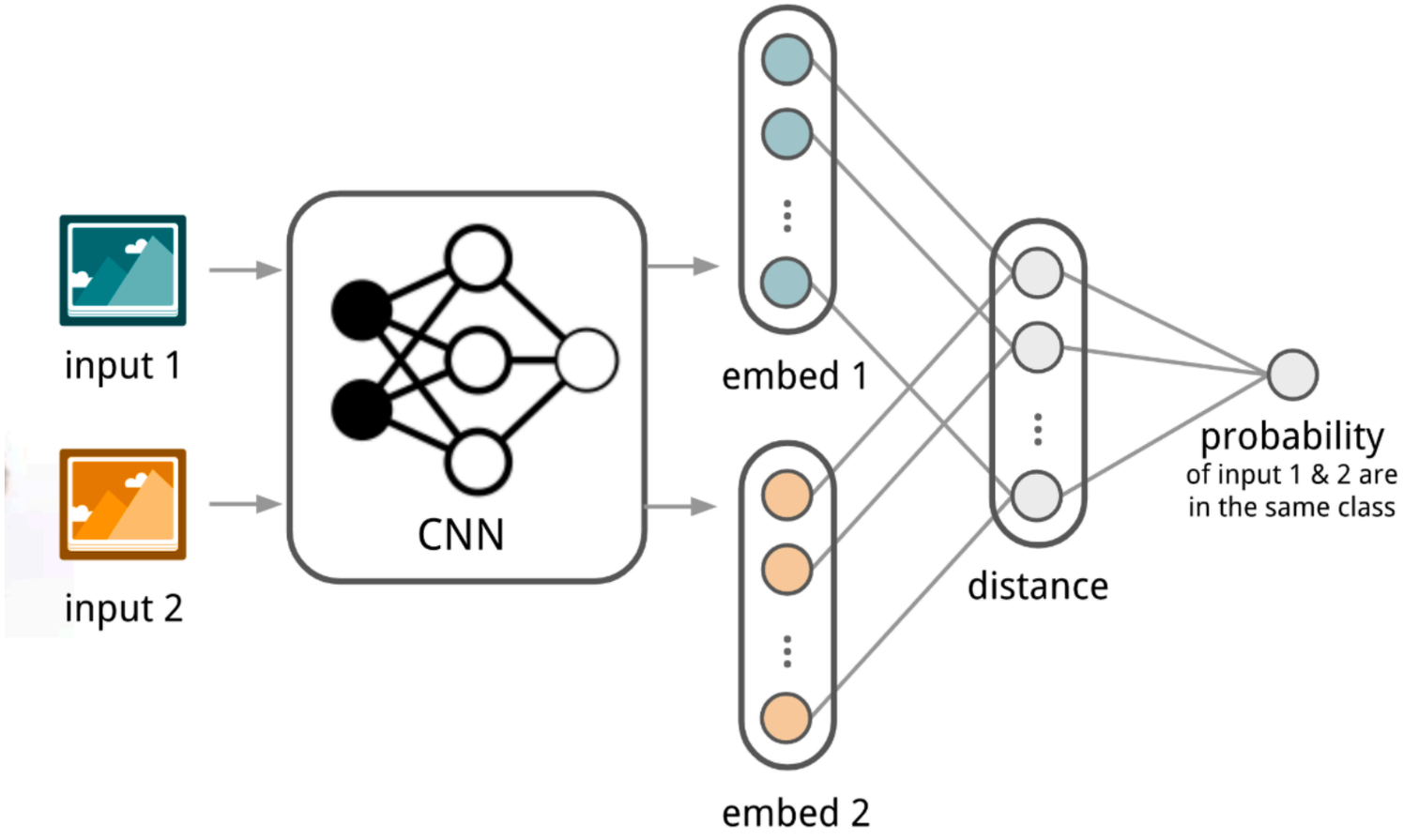
1. Metric-Based

- Similar ideas to nearest neighborhoods algorithm
- $p_\theta(y \mid x, S) = \sum_{(x_i, y_i) \in S} k_\theta(x, x_i) y_i$, where k_θ is the kernel function

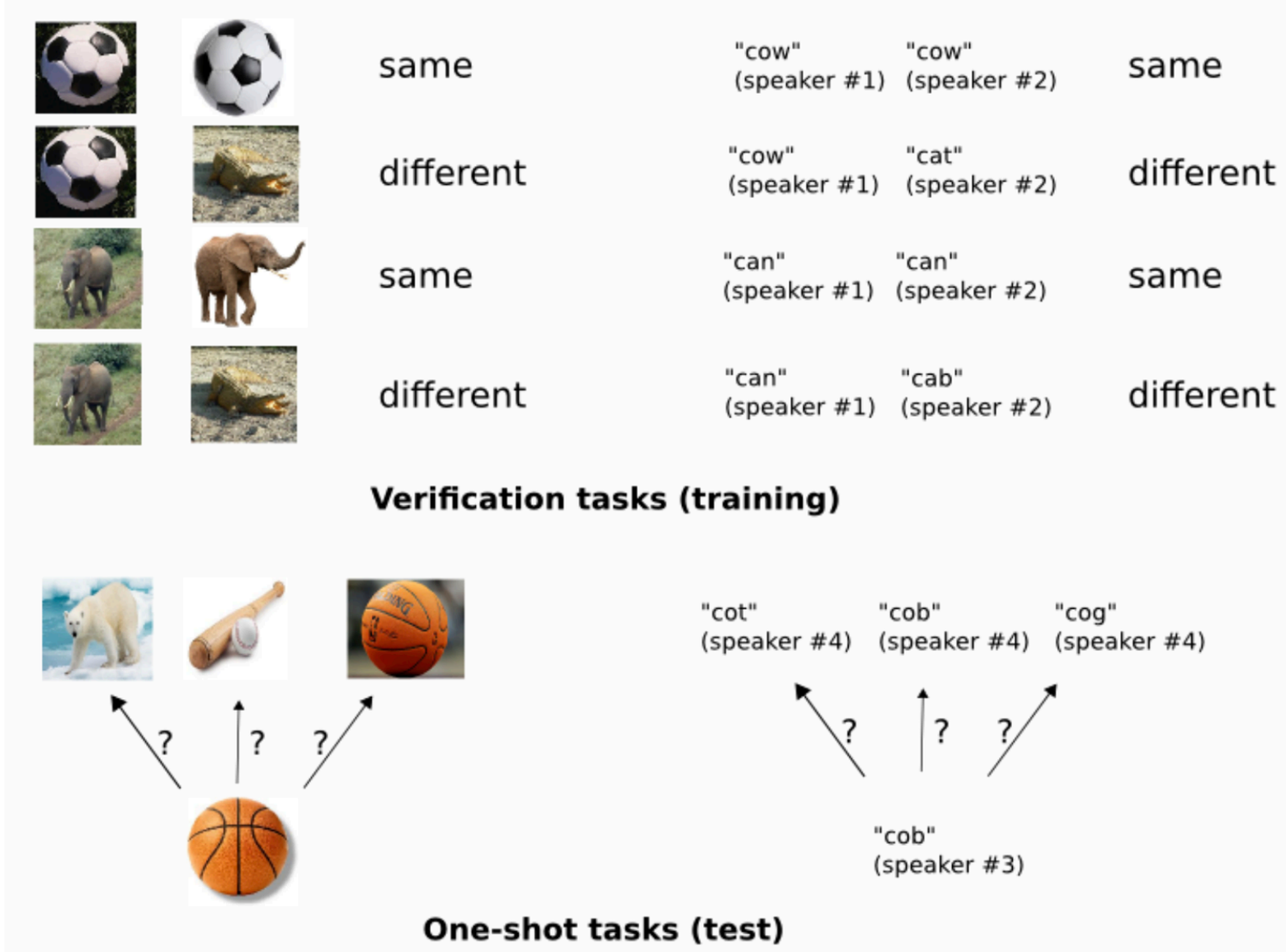
- Siamese Neural Networks for One-shot Image Recognition, ICML 2015
- Learning to Compare: Relation Network for Few-Shot Learning, CVPR 2018
- Matching Network for One-Shot Learning, NIPS 2016
- Prototypical Networks for Few-Shot Learning, NeurIPS 2017
- Few-Shot Learning with Graph Neural Networks, ICLR 2018

Siamese Neural Network

- Few-Shot Learning
- Twin network
- L1-distance as the metric



Siamese Neural Network



Relation Network

- Few-Shot Learning
- Similar to Siamese Network
- Difference: concatenation and CNN as the relation module

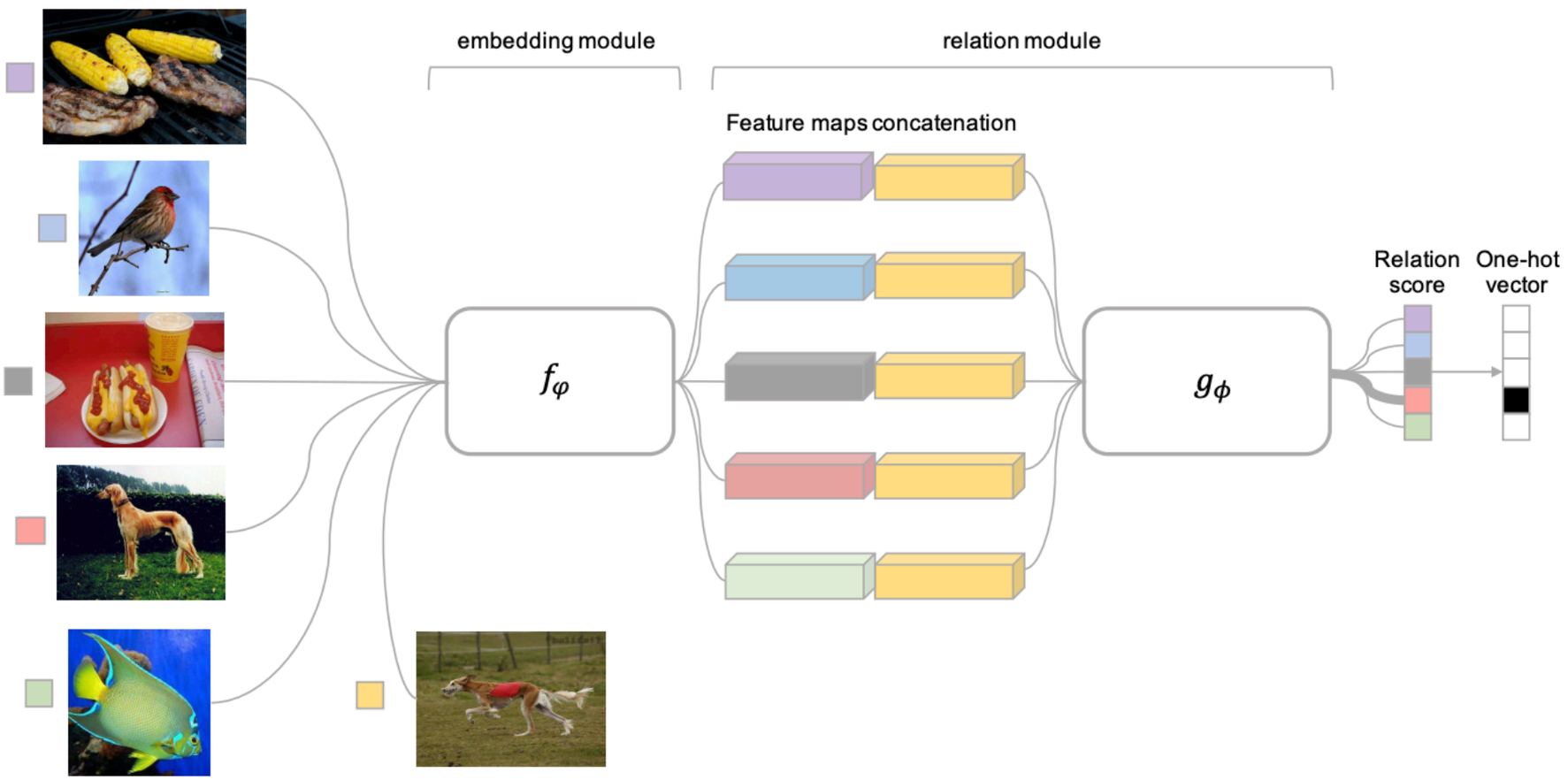


Figure 1: Relation Network architecture for a 5-way 1-shot problem with one query example.

Matching Network

- Given a training set (k samples per class): $S = \{x_i, y_i\}_{i=1}^{k}$
- Goal: $P(\hat{y} \mid \hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i = \sum_{i=1}^{k} \frac{\exp[cosine(f(\hat{x}), g(x_i))]}{\sum_{j=1}^{k} \exp[cosine(f(\hat{x}), g(x_j))]} y_i$
  - Two embedding methods are tested for f, g.

- Episodic Training
  - Support Set (C-Way K-Shot)

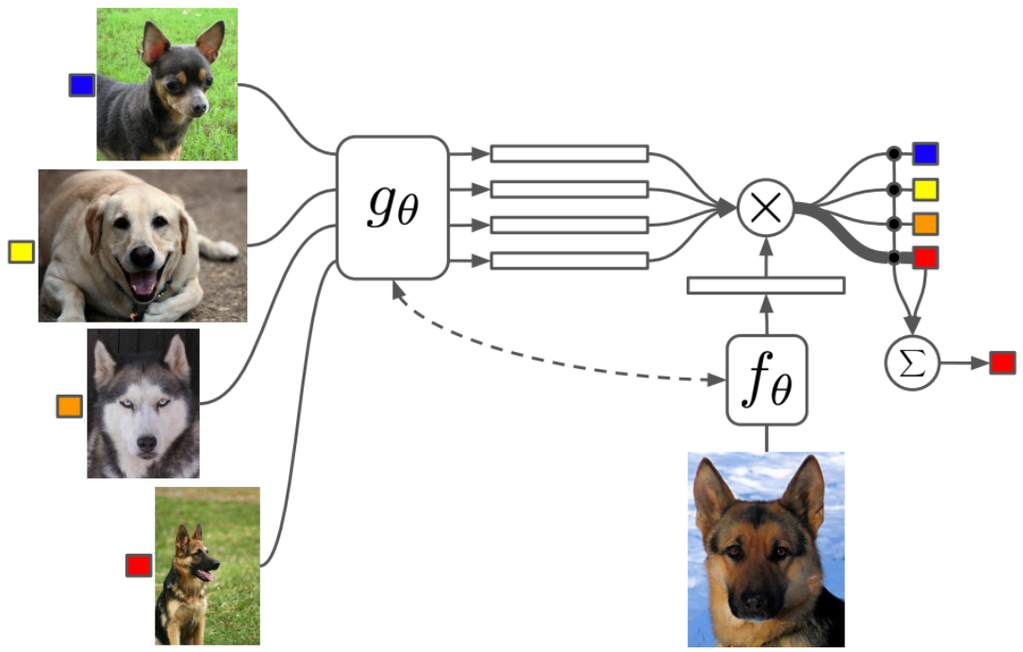


Figure 1: Matching Networks architecture

Matching Network

- **Simple Embedding**: with some CNN model and $f = g$
- **Full Context Embedding**:
  - $g(x_i)$ applies **bidirectional LSTM**
  - $f(\hat{x})$ applies **attention-LSTM**

1. First encodes through CNN to get $f'(\hat{x})$
2. Then an attention-LSTM is trained with a read attention over the full support set S

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

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

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

where $a(h_{k-1}, g(x_i)) = \exp\{h_{k-1}^T g(x_i)\} / \sum_{j=1}^{|S|} \exp\{h_{k-1}^T g(x_j)\}$

3. Finally $f(x) = h_K$, where K is # of read.

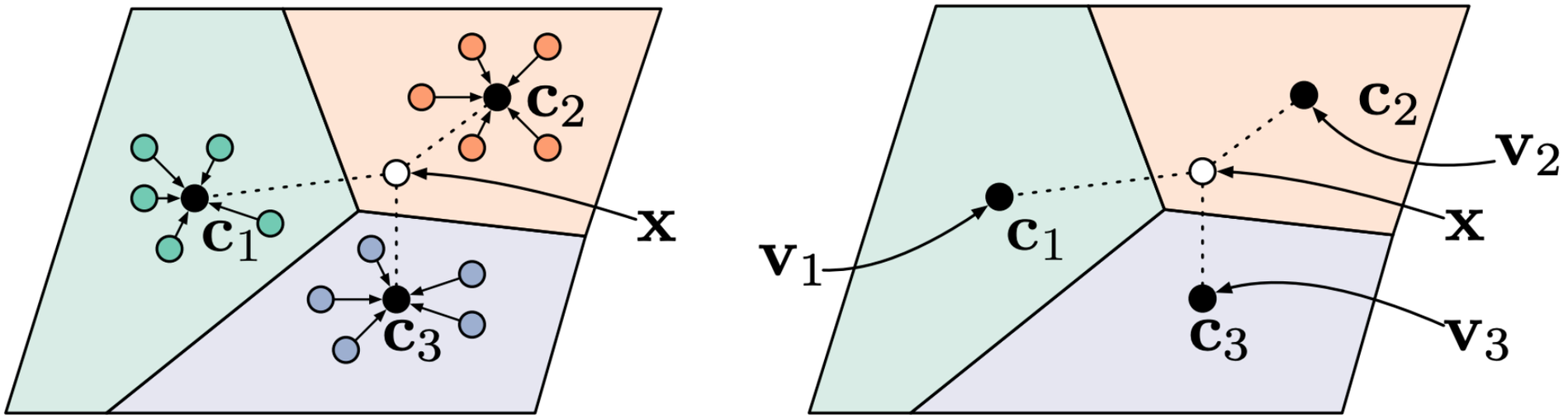
Prototypical Network

- For each class:
  - Sample a support set

$$c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i)$$

  - Sample a query set

$$p(y = k \mid x) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_{k'} \exp(-d(f_\phi(x), c_{k'}))}$$



(a) Few-shot

(b) Zero-shot

Prototypical Network

Algorithm 1 Training episode loss computation for Prototypical Networks. N is the number of examples in the training set, K is the number of classes in the training set, $N_C \leq K$ is the number of classes per episode, N_S is the number of support examples per class, N_Q is the number of query examples per class. RANDOMSAMPLE(S, N) denotes a set of N elements chosen uniformly at random from set S, without replacement.

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.

$V \leftarrow$ RANDOMSAMPLE$(\{1, \ldots, K\}, N_C)$ ▷ Select class indices for episode
for k in $\{1, \ldots, N_C\}$ **do**
  $S_k \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_{V_k}, N_S)$ ▷ Select support examples
  $Q_k \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_{V_k} \setminus S_k, N_Q)$ ▷ Select query examples
  $\mathbf{c}_k \leftarrow \frac{1}{N_C} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_{\boldsymbol{\phi}}(\mathbf{x}_i)$ ▷ Compute prototype from support examples
end for
$J \leftarrow 0$ ▷ Initialize loss
for k in $\{1, \ldots, N_C\}$ **do**
  for $(\mathbf{x}, y)$ in Q_k **do**
    $J \leftarrow J + \frac{1}{N_C N_Q} \left[d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) + \log \sum_{k'} \exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_{k'})) \right]$ ▷ Update loss
  end for
end for

Prototypical Network

- When viewed as a clustering algorithm, then the Bregman divergences can achieve the minimum distance to the center point in S

  $$d_\phi(z, z') = \phi(z) - \phi(z') - (z - z')^T \nabla_\phi(z')$$

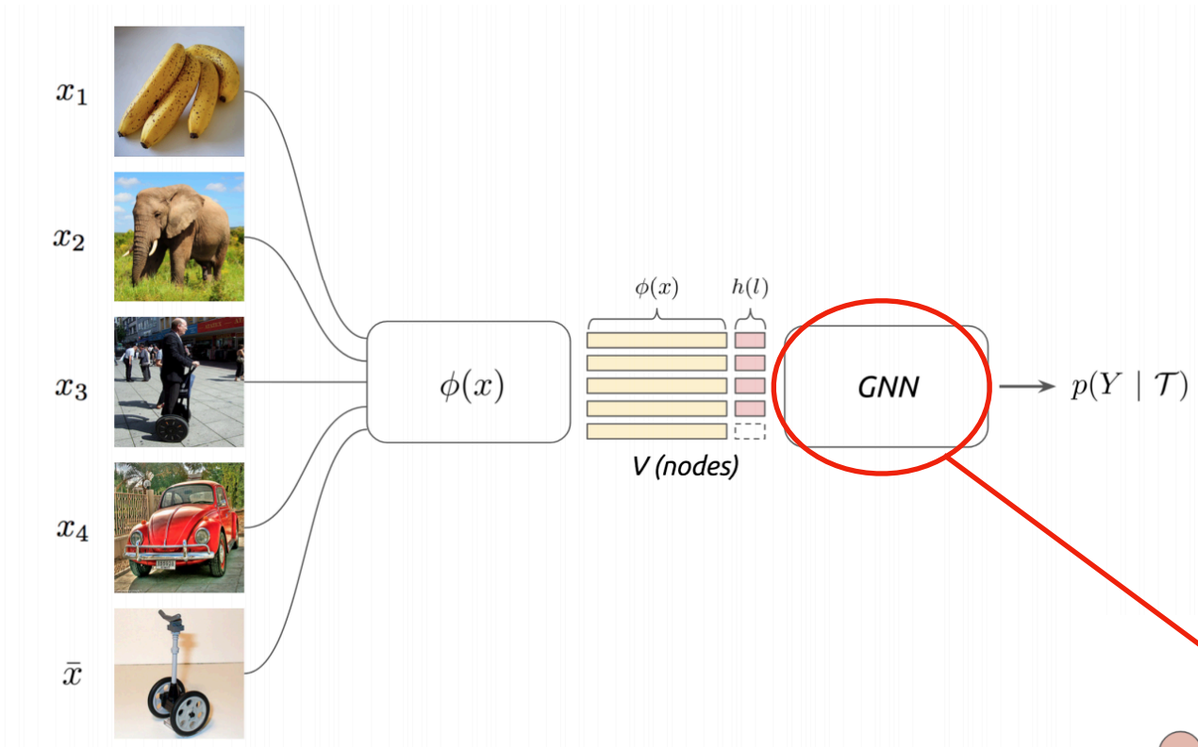
- Viewed as the linear regression when the Euclidean distance is used.
- Comparison between Matching Network & Prototypical Network:
  - equal in the one-shot learning, not in the K-shot learning
  - Matching Network:

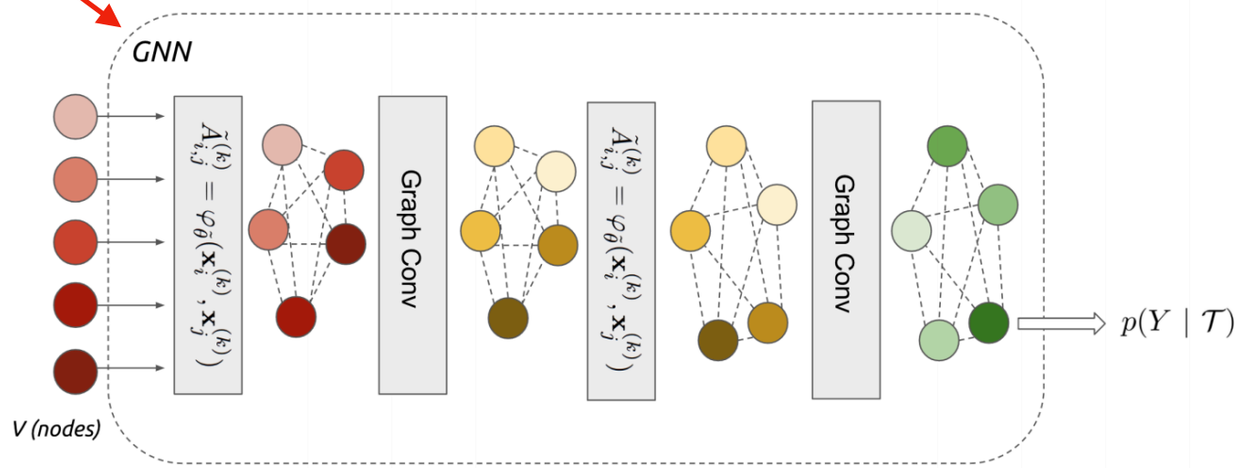
    $$P(\hat{y} \mid \hat{x}) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i = \sum_{i=1}^{k} \frac{\exp[cosine(f(\hat{x}), g(x_i))]}{\sum_{j=1}^{k} \exp[cosine(f(\hat{x}), g(x_j))]} \ y_i$$

  - Prototypical Network:

    $$p(y = k \mid x) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_{k'} \exp(-d(f_\phi(x), c_{k'}))}$$

Meta GNN


x_1
x_2
x_3
x_4
$\bar{x}$
$\phi(x)$
$\phi(x)$
$h(l)$
V (nodes)
GNN
$p(Y \mid \mathcal{T})$

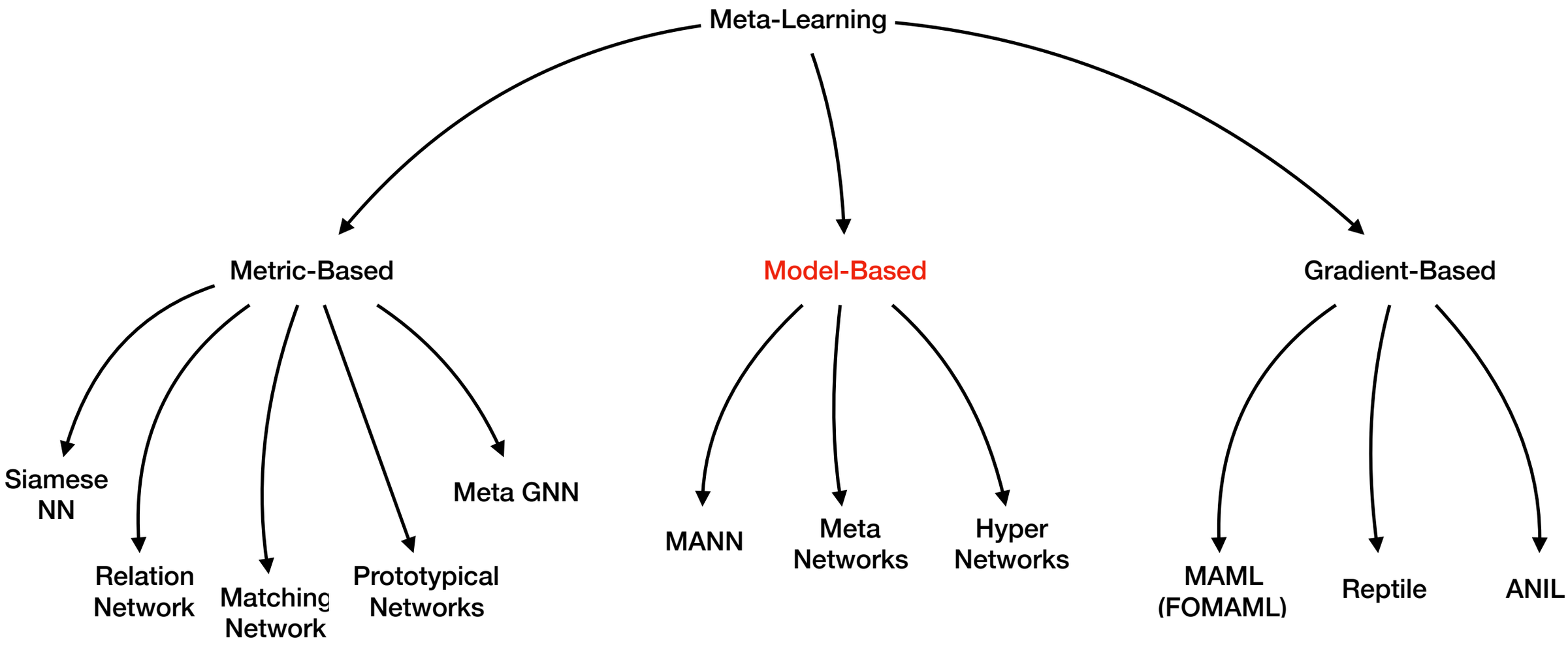

GNN
V (nodes)
$\tilde{A}_{i,j}^{(k)} = \varphi_\theta(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)})$
Graph Conv
$\tilde{A}_{i,j}^{(k)} = \varphi_\theta(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)})$
Graph Conv
$p(Y \mid \mathcal{T})$

Meta GNN

- For the k-th layer:
  - $x_i^k = \text{GCN}(x^{k-1})$
  - $A_{i,j}^k = \phi(x_i^k, x_j^k) = \text{MLP}(abs\,|\,x_i^k - x_j^k\,|\,)$

Metric-Based

- Comments:
  - Highly depends on the metric function.
  - Robustness: more troublesome when the new task diverges from the source tasks.


Meta-Learning
Metric-Based
Model-Based
Gradient-Based
Siamese NN
Relation Network
Matching Network
Prototypical Networks
Meta GNN
MANN
Meta Networks
Hyper Networks
MAML (FOMAML)
Reptile
ANIL

2. Model-Based

- Goal: to learn a model f_θ
- Solution: learning another model to parameterize f_θ

2. Model-Based

- Goal: to learn a base model f_θ
- Solution: learning a meta model to parameterize f_θ

2. Model-Based

- Goal: to learn a base model f_θ
- Solution: learning a meta model to parameterize f_θ

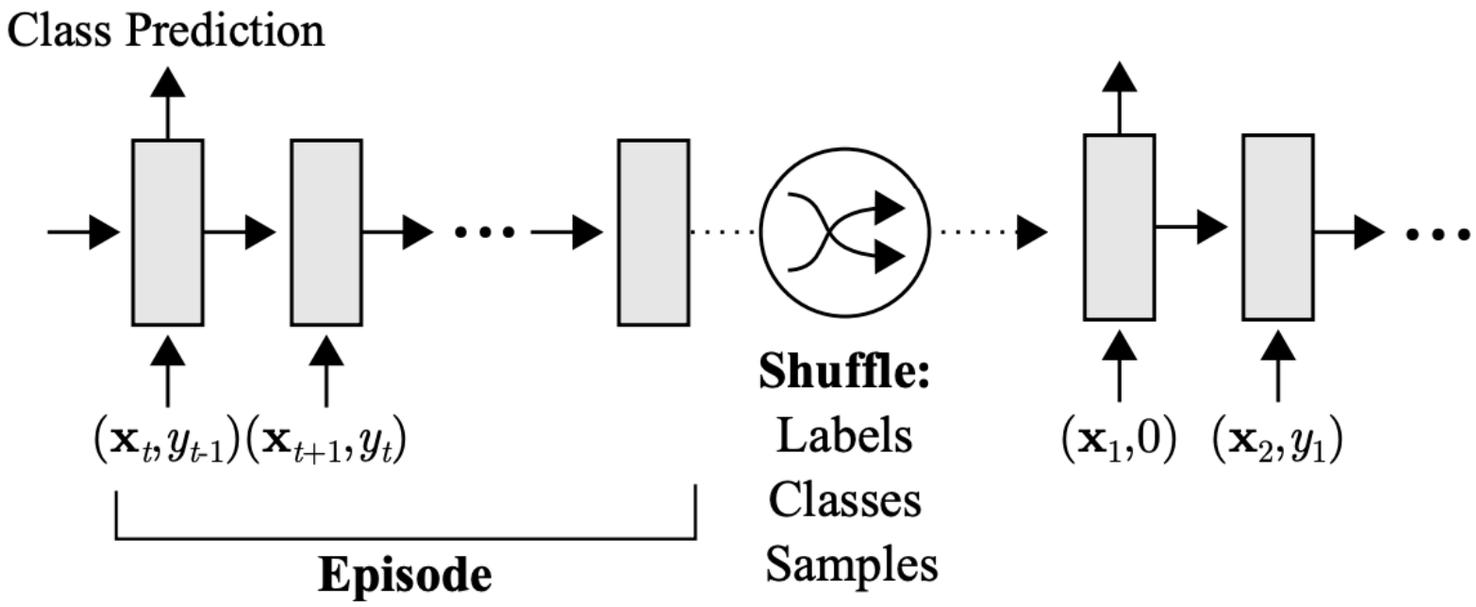
2. Model-Based

- Goal: to learn a base model f_θ
- Solution: learning a meta model to parameterize f_θ

- Meta-Learning with Memory-Augmented Neural Networks, ICML 2016
- Meta Networks, ICML 2017
- HyperNetworks, ArXiv 2016

Memory-Augmented Neural Networks (MANN)

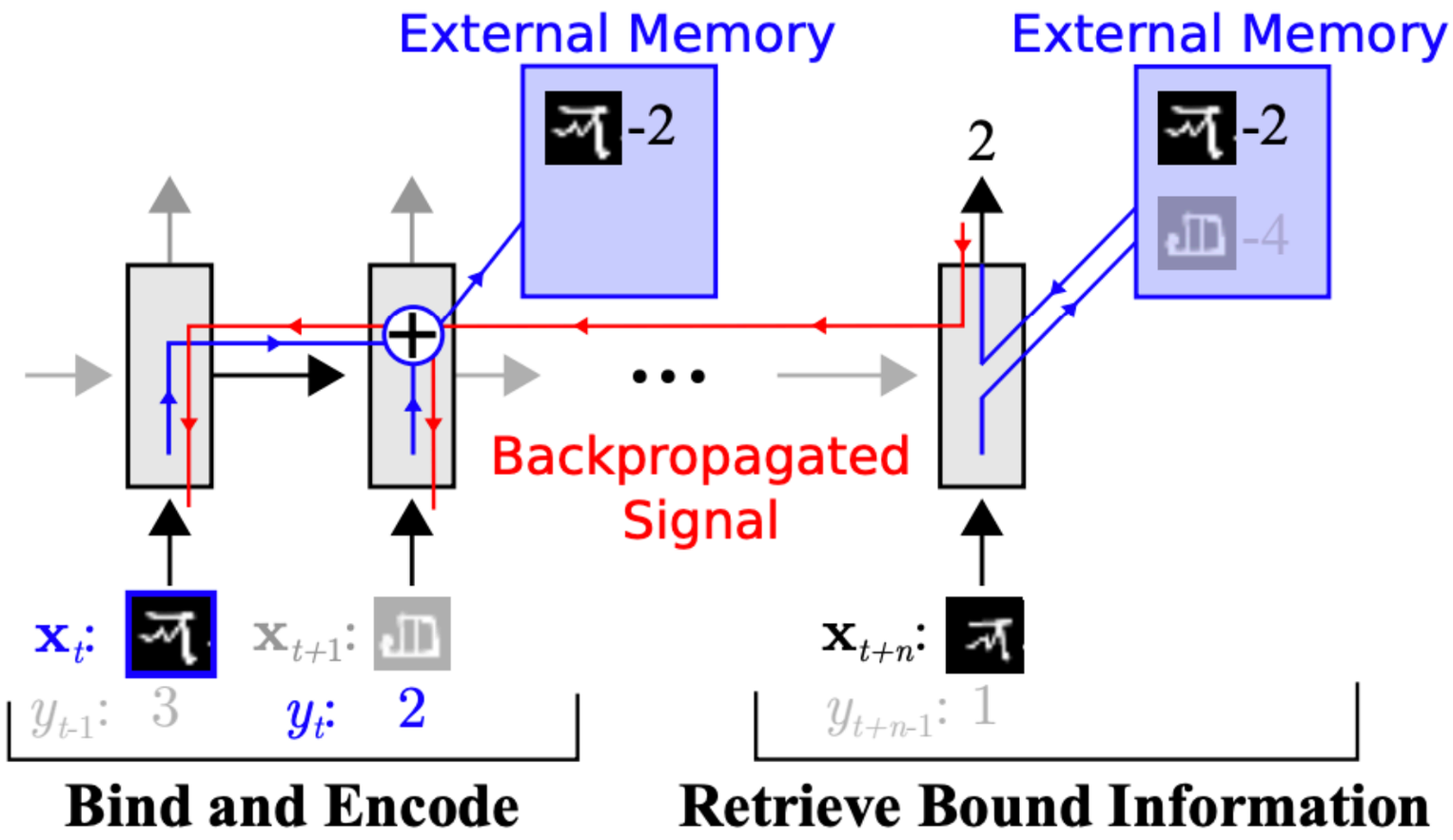
- Basic idea (Neural Turning Machine):
  - Store the useful information of the new task using an external memory.
  - The true label of the last time step is used.
  - External memory.



(a) Task setup

Memory-Augmented Neural Networks (MANN)

- Example:



(b) Network strategy

Addressing Mechanism

- key vector at step t k_t is generated from input x_t, memory matrix at step t is M_t, memory at step t is r_t
- read weights w_t^r, usage weights w_t^u, write weights w_t^w
- Read

$$w_t^r(i) = \text{softmax}\left(\exp((\frac{k_t M_t(i)}{\|k_t\|\|M_t(i)\|}))\right)$$

$$r_t = \sum_{i=1}^{N} w_t^r M_t(i)$$

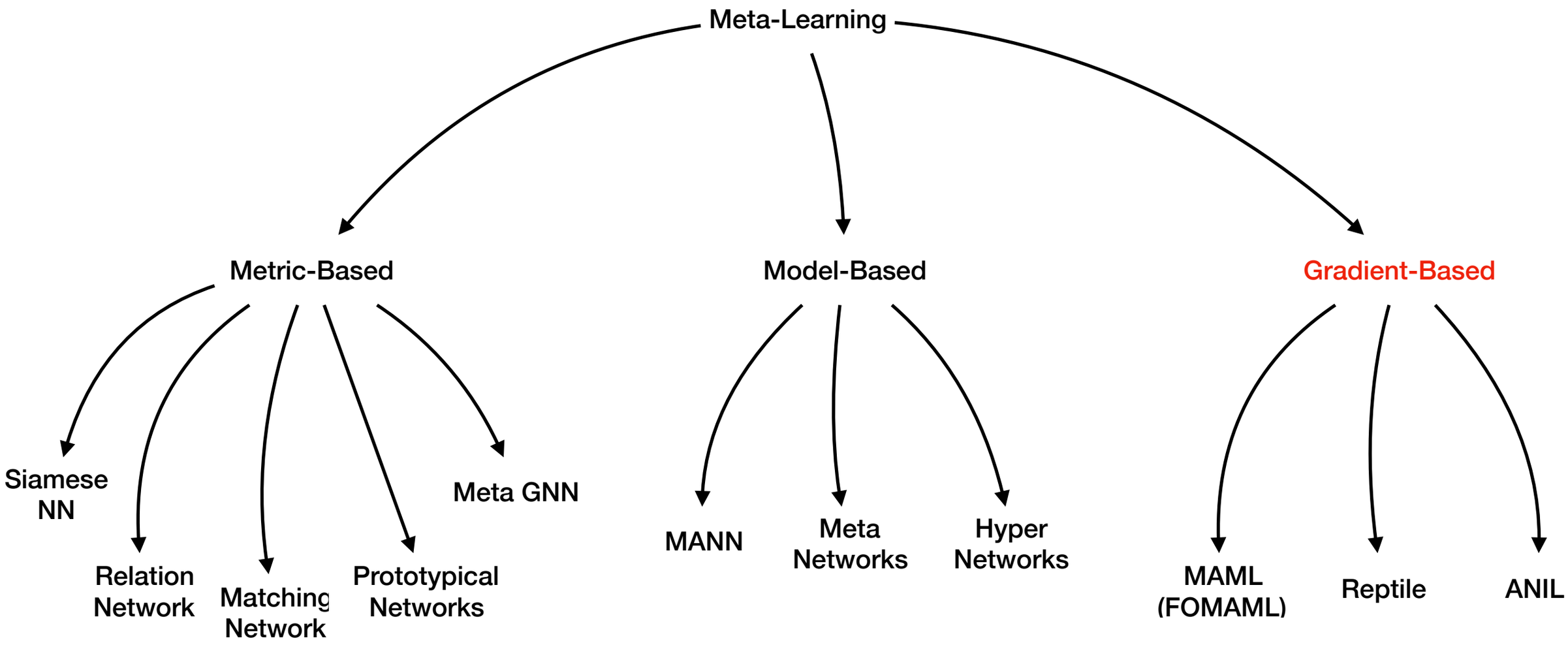
- Write (Least Recently Used Access, LRUA)

$$w_t^u = \gamma w_{t-1}^u + w_t^r + w_t^w$$

$$w_t^w = \sigma(\alpha) w_{t-1}^r + (1 - \sigma(\alpha)) w_{t-1}^{lu}$$

$$w_t^{ul} = \begin{cases} 0, \text{ if } w_t^u(i) > m(w_t^u, n) \\ 1, \text{ otherwise} \end{cases}$$, where $m(w_t^u, n)$ is the n-th smallest element in vector w_t^u

$$M_t(i) = M_{t-1}(i) + w_t^w(i) k_t, \forall i$$


Meta-Learning
Metric-Based
Model-Based
Gradient-Based
Siamese NN
Relation Network
Matching Network
Prototypical Networks
Meta GNN
MANN
Meta Networks
Hyper Networks
MAML (FOMAML)
Reptile
ANIL

3. Gradient-Based

Model-Based:

- Goal: to learn a base model f_θ
- Solution: learning a meta model to parameterize f_θ

3. Gradient-Based

Model-Based:

- Goal: to learn a base model f_θ
- Solution: learning a meta model to parameterize f_θ

Gradient-Based:

- Goal: to learn a base model f_θ
- Solution: learning to parameterize f_θ without a meta model

3. Gradient-Based

- Learning to learn with Gradients

- MAML (Model-Agnostic Meta Learning） & FOMAML, ICML 2017
- Reptile, ArXiv 2018
- ANIL (Almost No Inner Loop), ICLR 2020

MAML

- Model-Agnostic Meta-Learning (MAML)
- Motivation
  - find a model parameter that are sensitive to changes in the task
  - small changes in the parameters can get large improvements

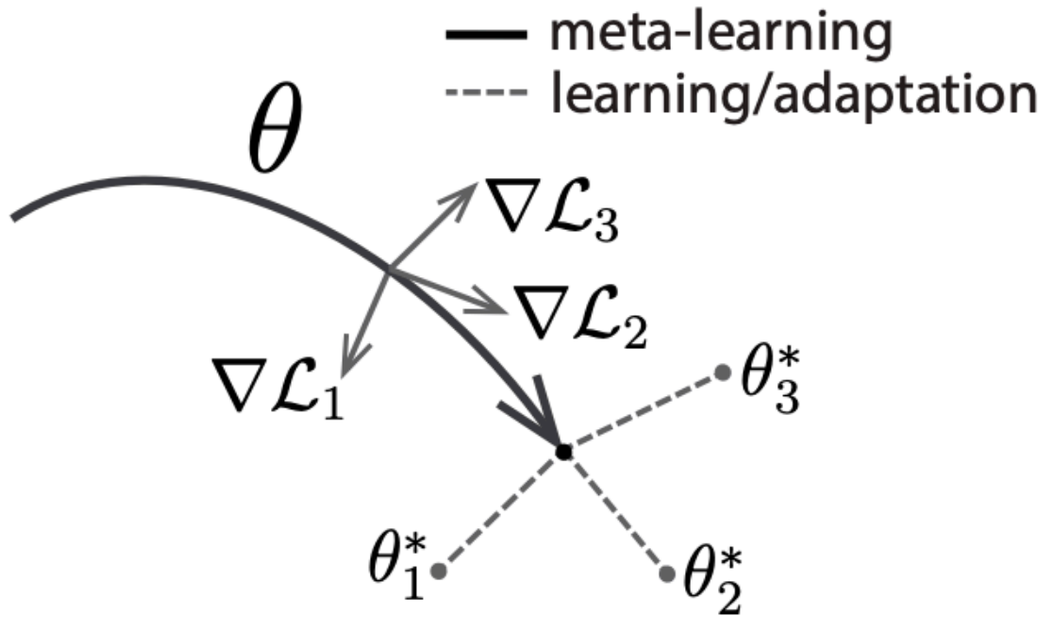


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

MAML

- Outer loop:
  - Inner loop:
    - Sample batch of tasks τ_i
    - Sample K samples
  - Meta-object: $\min_{\theta} \sum_{\tau_i \sim P(\tau)} \ell_{\tau_i}(f_{\theta_i'}) = \sum_{\tau_i \sim P(\tau)} \ell_{\tau_i}(f_{\theta - \alpha \nabla_\theta \ell_{\tau_i}(f_\theta)})$
  - SGD: $\theta = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta_i'}) = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta - \alpha \nabla_\theta \ell_{\tau_i}(f_\theta)})$

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**

FOMAML

- Involves a gradient through a gradient:

$$\theta = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta - \alpha \nabla_\theta \ell_{\tau_i}(f_\theta)})$$

- First-order approximation, A.K.A. first-order MAML (FOMAML)
  - Omit the second-order derivatives
  - Still compute the meta-gradient at the post-update parameter θ_i'
  - $$\theta = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta_i'})$$
  - Almost the same performance, but ~33% faster

- Notice: this meta-objective is multi-task learning.

MAML

- Outer loop:
  - Inner loop:
    - Sample batch of tasks τ_i
    - Sample K samples
    - Meta-object: $\min_{\theta} \sum_{\tau_i \sim P(\tau)} \ell_{\tau_i}(f_{\theta_i'}) = \sum_{\tau_i \sim P(\tau)} \ell_{\tau_i}(f_{\theta - \alpha \nabla_\theta \ell_{\tau_i}(f_\theta)})$
    - SGD: $\theta = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta_i'}) = \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta - \alpha \nabla_\theta \ell_{\tau_i}(f_\theta)})$

FOMAML

- Outer loop:
  - Inner loop:
    - Sample batch of tasks τ_i
    - Sample K samples
    - Meta-object: $\min_{\theta} \sum_{\tau_i \sim P(\tau)} \ell_{\tau_i}(f_{\theta_i'}) = \sum_{\tau_i \sim P(\tau)} \ell_{\tau_i}(f_{\theta - \alpha \nabla_{\theta} \ell_{\tau_i}(f_{\theta})})$
    - SGD: $\theta' = \theta - \alpha \nabla_{\theta} \ell_{\tau_i} f(\theta)$
  - SGD: $\theta = \theta - \beta \nabla_{\theta} \sum_{\tau_i \sim p(\tau)} \ell_{\tau_i}(f_{\theta_i'})$

Reptile

- Same motivation:
  - pre-training: learn a initialization
  - fine-tuning: able to **quickly** be adapted on new tasks

Reptile

- For each iteration, do:
  - Sample task τ
  - Get the corresponding loss ℓ_τ
  - Compute $\tilde{\theta} = U_\tau^k(\theta)$, with k steps of SGD/Adam
  - Update $\theta = \theta + \epsilon(\tilde{\theta} - \theta)$ or $\theta = \theta + \epsilon \frac{1}{n} \sum_{i=1}^{n} (\tilde{\theta}_i - \theta)$

Reptile

- If $k = 1$, Reptile is similar to $\min \mathbb{E}_\tau[L_\tau]$

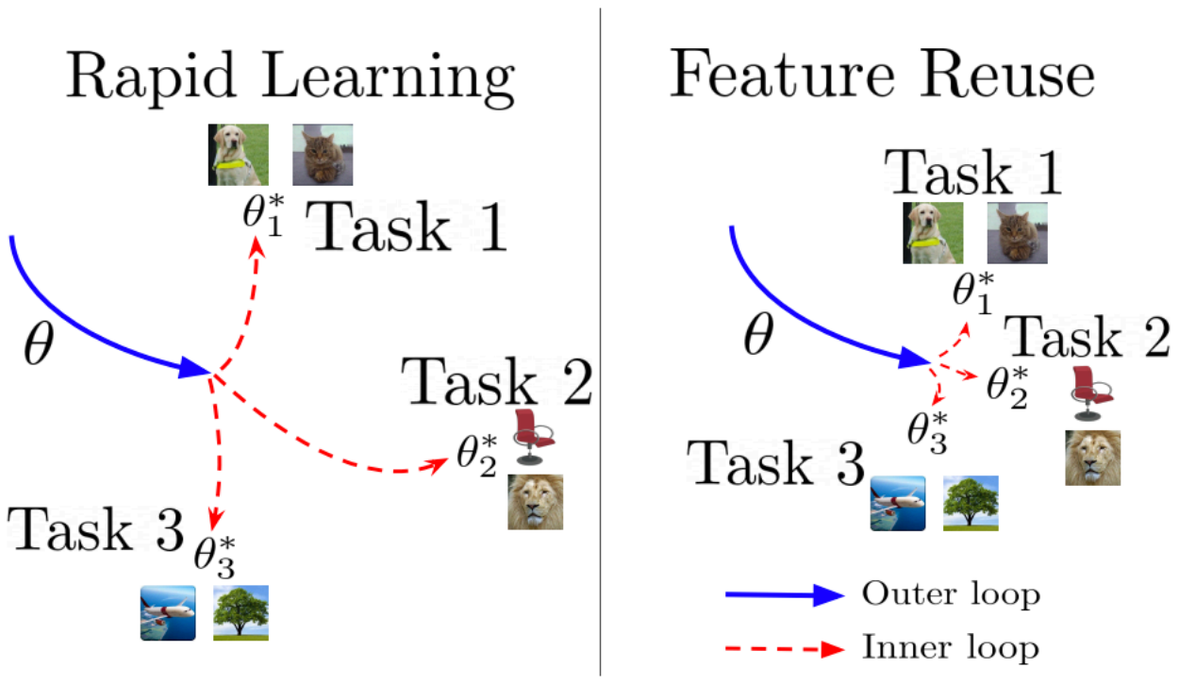
$$g_{Reptile,k=1} = \theta - \tilde{\theta} = \theta - U_{\tau,A}(\theta) = \theta - (\theta - \nabla_\theta L_{\tau,A}(\theta)) = \nabla_\theta L_{\tau,A}(\theta)$$

- If $k > 1$, Reptile diverges from $\min \mathbb{E}_\tau[L_\tau]$

$$\theta - U_{\tau,A}(\theta) \neq \theta - (\theta - \nabla_\theta L_{\tau,A}(\theta))$$

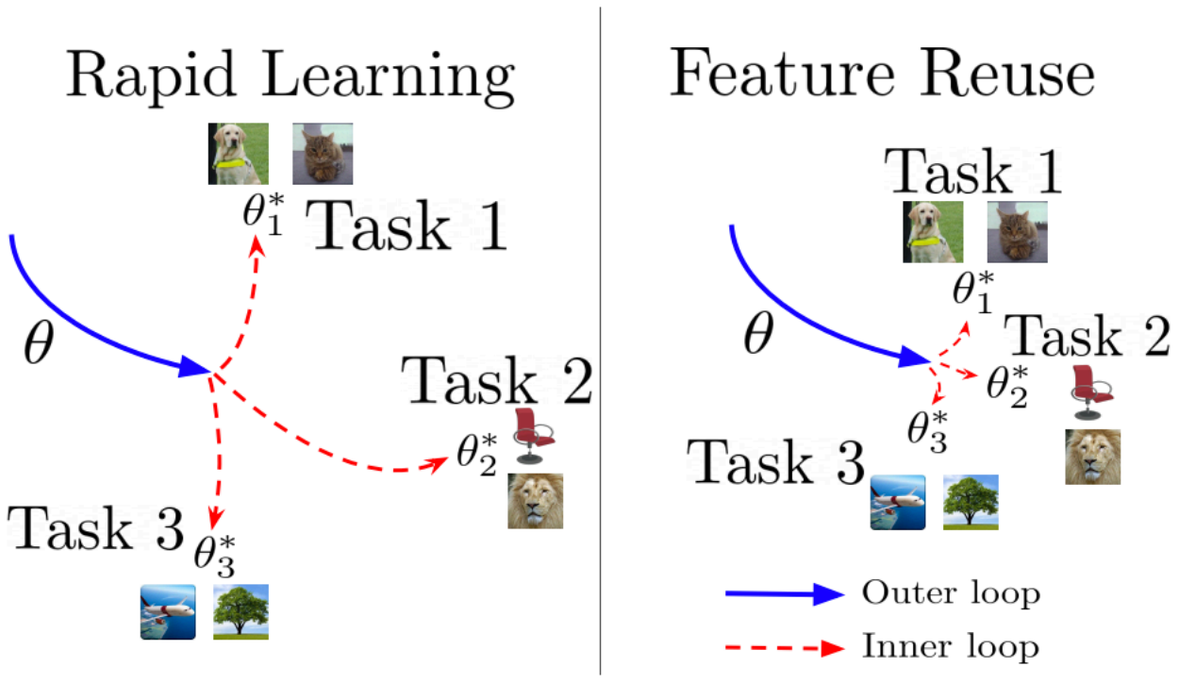
ANIL

- ANIL (Almost No Inner Loop)
- The reason why MAML works: rapid learning or feature reuse



ANIL

- ANIL (Almost No Inner Loop)
- The reason why MAML works: rapid learning or feature reuse



ANIL

- ANIL: Only update the head (last layer) in the inner loop

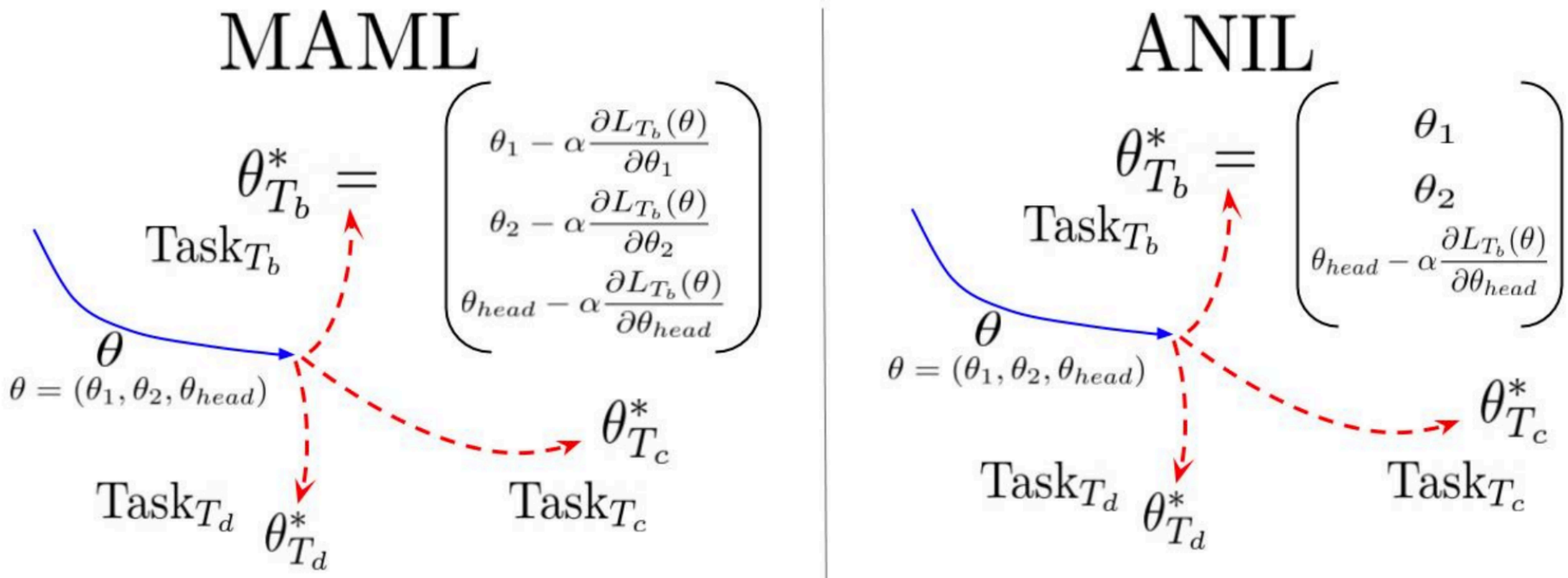


Figure 4: **Schematic of MAML and ANIL algorithms.** The difference between the MAML and ANIL algorithms: in MAML (left), the inner loop (task-specific) gradient updates are applied to all parameters θ, which are initialized with the meta-initialization from the outer loop. In ANIL (right), only the parameters corresponding to the network head θ_{head} are updated by the inner loop, during training **and** testing.

Meta-Learning on Drug Discovery

- Meta-Learning Initialization for Low-Resource Drug Discovery, ArXiv 2020
  - Applied MAML, FOMAML, ANIL on drug data

Thank You

- Questions?