Neural Relational Inference

Presenter: Zhe Wang

https://qdata.github.io/deep2Read

Zhe Wang

201909

Neural relational inference for interacting systems

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, Richard Zemel

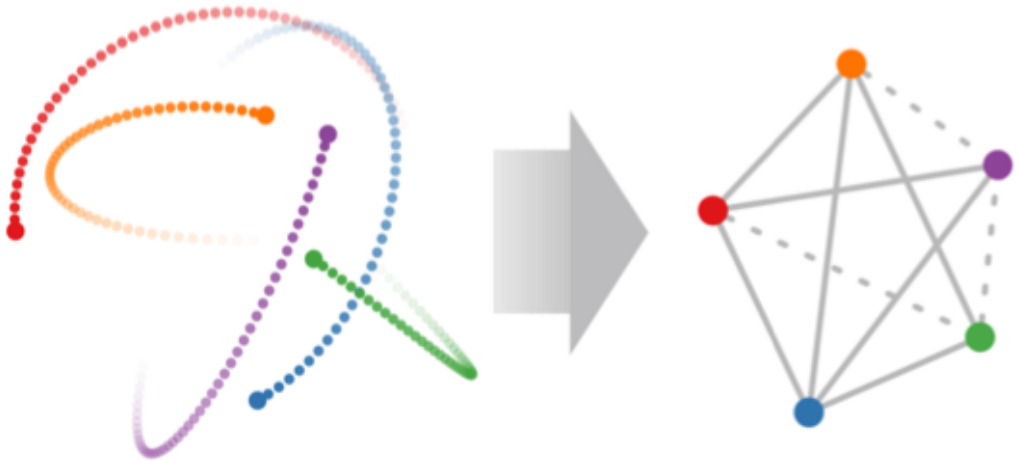
ICML 2018

Background

Interacting system, such as charged particles and particles connected by springs, are prevalent in nature.

Given the interactions, they can be modeled as physical systems,and the future behaviours can be predicted via ODE.

The goal is to predict the interaction from observable trajectories only.



Task

Task: inferring an **explicit interaction structure** while simultaneously learning the **dynamical model** of the interacting system in an **unsupervised** way.

Notations:

- x_i^t the feature vector of object v_i at time t, e.g. location and velocity.
- $x^t = \{x_1^t, , x_N^t\}$ the set of features of all N objects at time t
- $x_i = (x_i^1, \cdots, x_i^T)$ the trajectory of object i, where T is the total number of time steps.

Main idea: the structure is latent and will be recovered by variation inference.

NRI model is formalized as a VAE, the ELBO to be maximized is:

$$L = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p_\theta(z)]$$

Encoder: $q_\phi(z|x)$
Decoder: $p_\theta(x|z)$
Prior distribution:$p_\theta(z) = \prod_{i \neq j} p_\theta(z_{ij})$ is a factorized uniform distribution over edges types.

Encoder: Infer pairwise interaction types z_{ij} given observed trajectories $\mathrm{x} = (\mathrm{x}^1, \cdots, \mathrm{x}^T)$.

Model: GNN on the fully connected graph without self-loops.

Two types of modules: node embedding and edge embedding.

$$
\begin{aligned}
& & \mathbf{h}_j^1 &= f_{\text{emb}}(\mathbf{x}_j) \\
v \to e: & & \mathbf{h}_{(i,j)}^1 &= f_e^1([\mathbf{h}_i^1, \mathbf{h}_j^1]) \\
e \to v: & & \mathbf{h}_j^2 &= f_v^1\left(\textstyle\sum_{i \neq j} \mathbf{h}_{(i,j)}^1\right) \\
v \to e: & & \mathbf{h}_{(i,j)}^2 &= f_e^2([\mathbf{h}_i^2, \mathbf{h}_j^2])
\end{aligned}
$$

Posterior: $q_\phi(z_{ij}|x) = softmax(h^2_{(i,j)})$

Sampling: Gumbel-softmax.

$$z_{ij} = softmax((h^2_{(i,j)} + g)/\tau)$$

Decoder: GNN to do reconstruction.

$$
\begin{aligned}
v \to e: \quad \tilde{\mathbf{h}}_{(i,j)}^{t} &= \sum_{k} z_{ij,k} \tilde{f}_{e}^{k}([\mathbf{x}_i^t, \mathbf{x}_j^t]) \\
e \to v: \quad \boldsymbol{\mu}_j^{t+1} &= \mathbf{x}_j^t + \tilde{f}_v(\textstyle\sum_{i \neq j} \tilde{\mathbf{h}}_{(i,j)}^{t}) \\
p(\mathbf{x}_j^{t+1} | \mathbf{x}^t, \mathbf{z}) &= \mathcal{N}(\boldsymbol{\mu}_j^{t+1}, \sigma^2 \mathbf{I})
\end{aligned}
$$

$z_{ij,k}$ denotes the k-th element of the vector z_{ij}

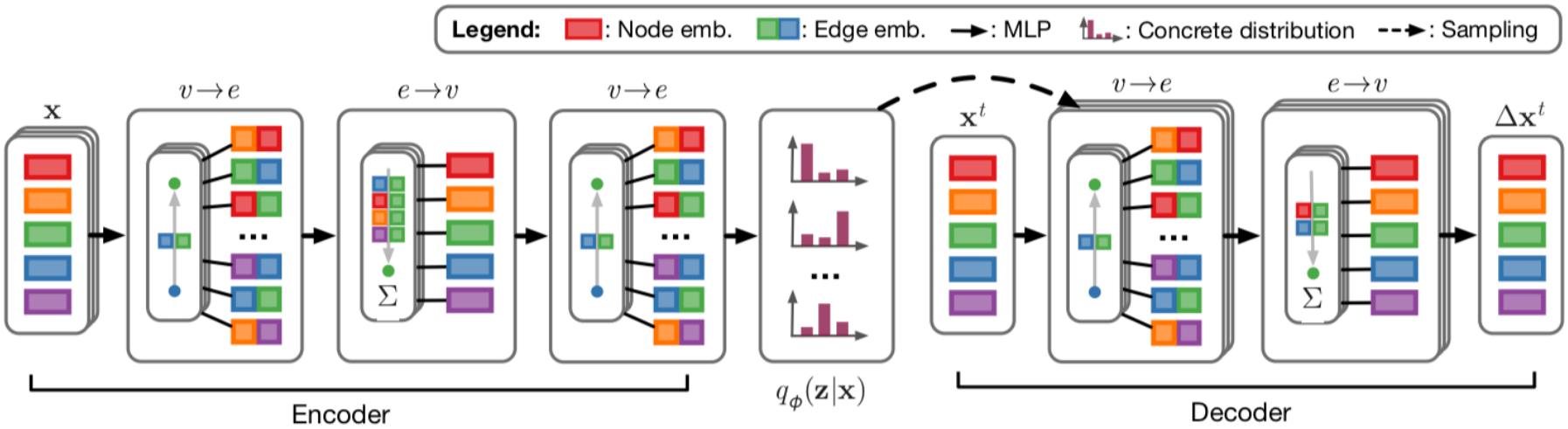
Training:
Given training example x, compute $q_\phi(z_{ij}|x)$, then we sample z_{ij} from the concrete reparameterizable approximation of $q_\phi(z_{ij}|x)$. We then run the decoder to compute $\mu_2, \cdots, \mu_T$.

rec error:

$$-\sum_{j}\sum_{t=2}^{T}\frac{||x_j^t-\mu_j^t||^2}{2\sigma^2}+c$$

KL reg:

$$\sum_{i\neq j}H(q_\phi(z_{ij}|x))+c$$



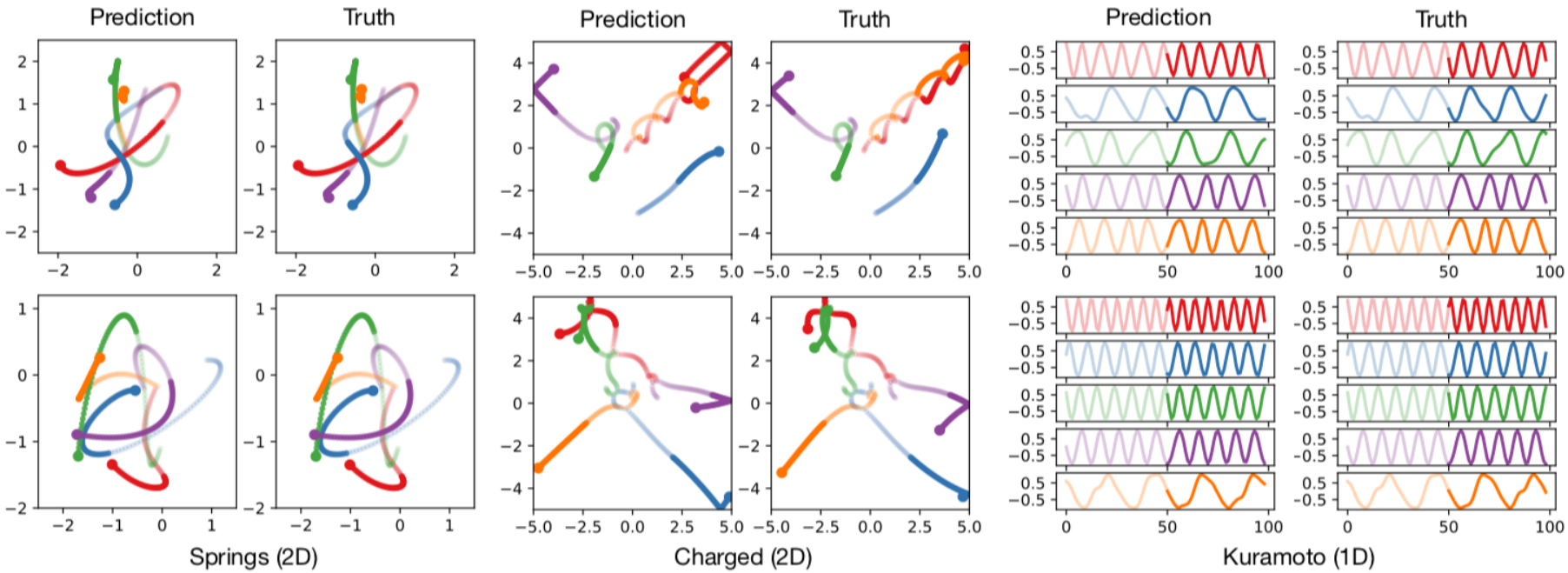


Figure 5. Trajectory predictions from a trained NRI model (unsupervised). Semi-transparent paths denote the first 49 time steps of ground-truth input to the model, from which the interaction graph is estimated. Solid paths denote self-conditioned model predictions.

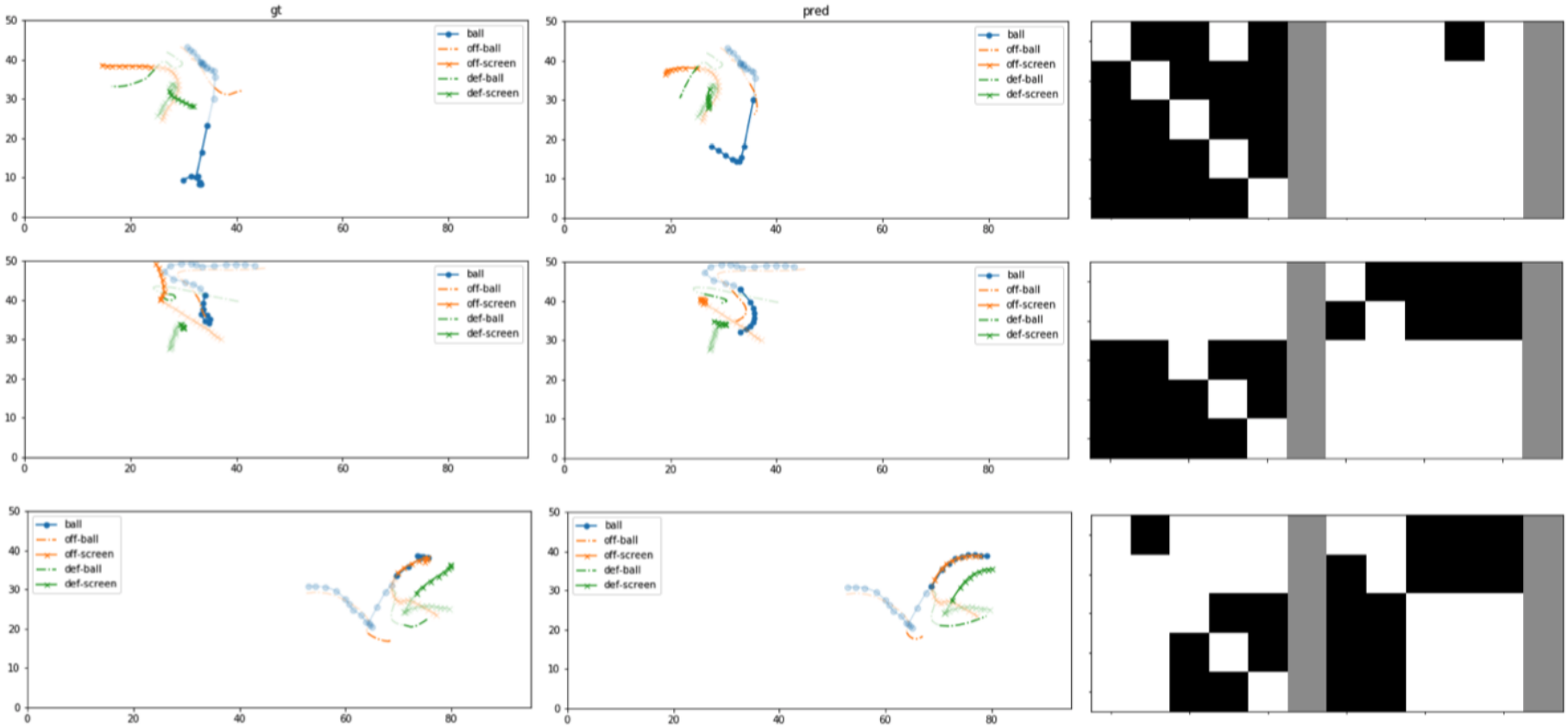


Figure 11. Visualization of NBA trajectories. *Left*: ground truth; *middle*: model prediction; *right*: sampled edges.

procedure BOUNCEGRAD($\mathbb{S}$, $D_1^{train}, \ldots, D_m^{train}, D_1^{test}, \ldots, D_m^{test}, \eta, T_0, \Delta_T, T_{end}$)
  $S_1, \ldots, S_m$ = random simple structures from $\mathbb{S}$; Θ = neural-network weight initialization
  for $T = T_0;\ T = T - \Delta_T;\ T < T_{end}$ **do**
    BOUNCE($S_1, \ldots, S_m, D_1^{train}, \ldots, D_m^{train}, T, \mathbb{S}, \Theta$)
    GRAD($\Theta, S_1, \ldots, S_m, D_1^{test}, \ldots, D_m^{test}, \eta$)

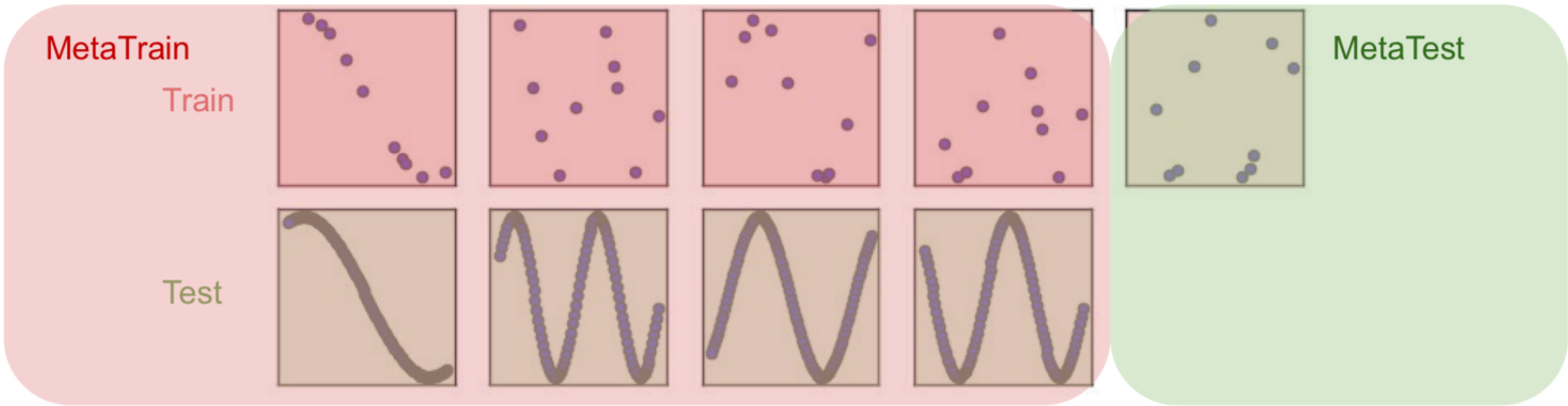
Neural Relational Inference with Fast Modular Meta-learning

Ferran Alet, Erica Weng, Tomás Lozano Pérez, Leslie Pack Kaelbling

NeurIPS 2019

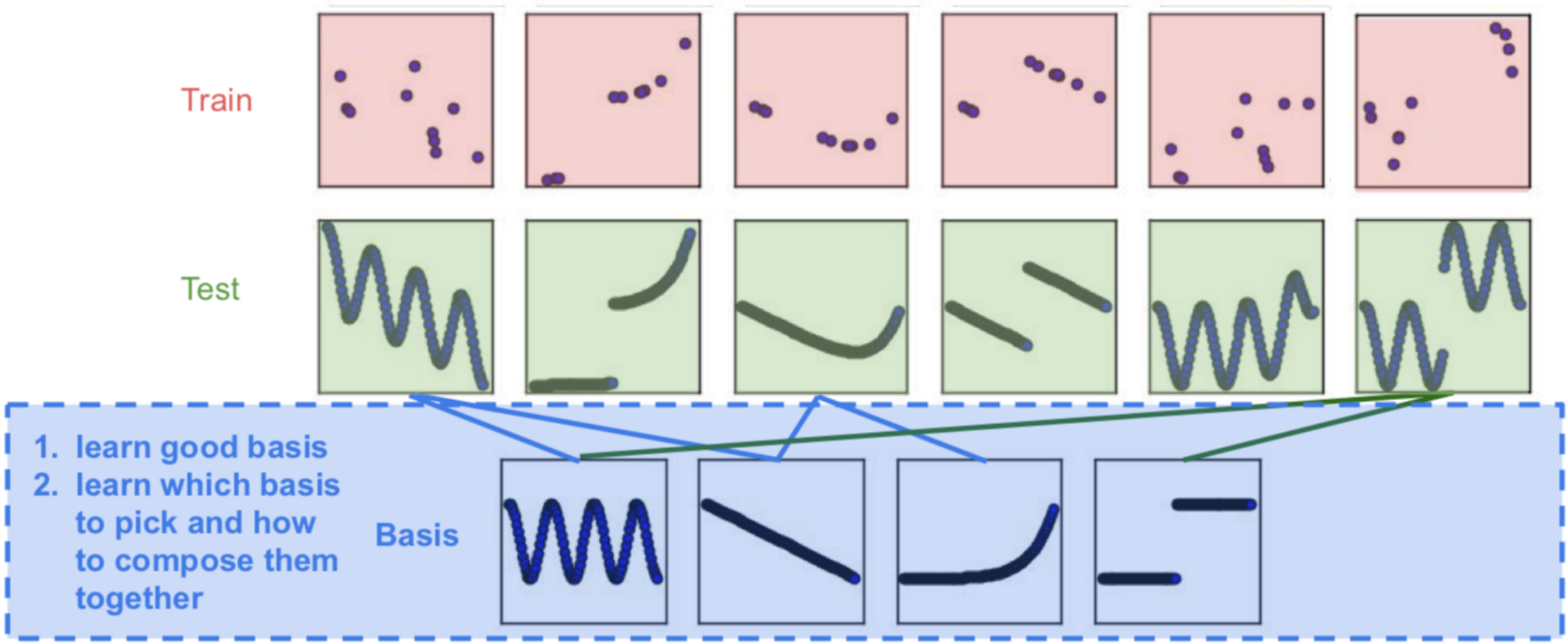
Meta-learning

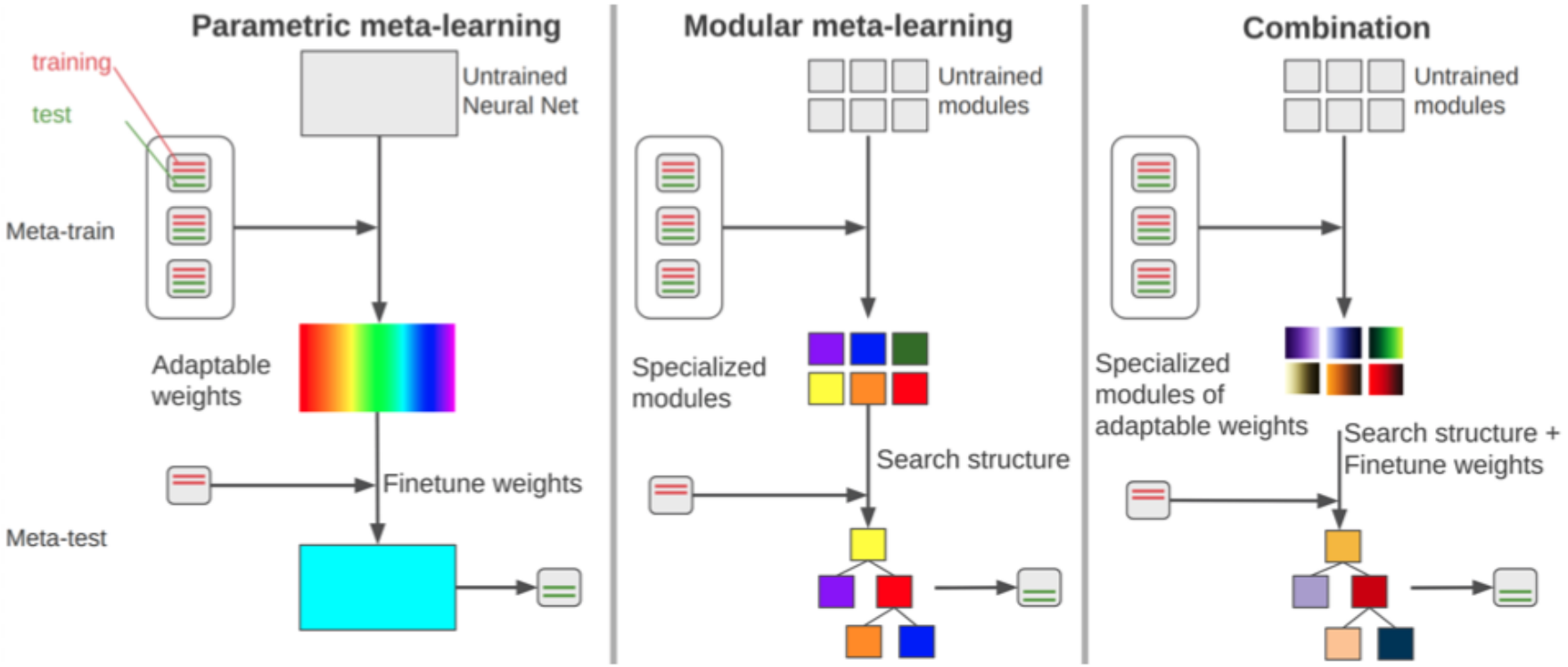
learns characteristics shared by similar tasks



Modular meta-learning

learns a *modular decomposition* of characteristics shared by similar tasks





Inner Loop: Learn how to compose basis functions for each individual task
Outer Loop: Optimize the parameters of all basis.

Composition schemes:

- sum
- attention based weighted sum
- concatenation
- A general function-composition tree, where the local modifications include both changing which f_i is used at each node, as well as adding or deleting nodes from the tree.

Meta testing phase:
All modules are fixed, we only need to find an optimal structure $S \in \mathsf{S}$ for the specific task according to the loss on the training set.

$$S^* = \arg\min_{s \in \mathsf{S}} (D^{train}_{meta_test}, \mathsf{S}, \theta)$$

Algorithm: simulated annealing.

procedure ONLINE($D^{train}_{meta-test}$, $\mathbb{S}$, Θ, T_0, Δ_T, T_{end})
  S = random simple structure from $\mathbb{S}$
  for $T = T_0;\ T = T - \Delta_T;\ T < T_{end}$ **do**
    $S' = \text{PROPOSE}_{\mathbb{S}}(S)$
    if ACCEPT($e(D, S', \Theta), e(D, S, \Theta), T$) **then** $S = S'$
  return S
procedure ACCEPT(v', v, T)
  return $v' < v$ or rand(0, 1) $< \exp\{(v - v')/T\}$

Meta training phase:
For each task, in the inner loop, find the optimal structure with simulated annealing on training set.
in the outer loop, optimize the loss on test set with respect to module parameters.

procedure BOUNCEGRAD($\mathbb{S}$, $D_1^{train}, \ldots, D_m^{train}, D_1^{test}, \ldots, D_m^{test}, \eta, T_0, \Delta_T, T_{end}$)
  $S_1, \ldots, S_m$ = random simple structures from $\mathbb{S}$; Θ = neural-network weight initialization
  for $T = T_0;\ T = T - \Delta_T;\ T < T_{end}$ **do**
    BOUNCE($S_1, \ldots, S_m, D_1^{train}, \ldots, D_m^{train}, T, \mathbb{S}, \Theta$)
    GRAD($\Theta, S_1, \ldots, S_m, D_1^{test}, \ldots, D_m^{test}, \eta$)

procedure BOUNCE($S_1, \ldots, S_m, D_1^{train}, \ldots, D_m^{train}, T, \mathbb{S}, \Theta$)
  for $j = 1 \ldots m$ **do**
    $S'_j = Propose_{\mathbb{S}}(S_j, \Theta)$
    if $Accept(e(D_j^{train}, S'_j, \Theta), e(D_j^{train}, S_j, \Theta), T)$ **then** $S_j = S'_j$

procedure GRAD($\Theta, S_1, \ldots, S_m, D_1^{test}, \ldots, D_m^{test}, \eta$)
  $\Delta = 0$
  for $j = 1 \ldots m$ **do**
    $(x, y) = \text{rand\_elt}(D_j^{test})$; $\Delta = \Delta + \nabla_\Theta L(S_{j\Theta}(x), y)$
  $\Theta = \Theta - \eta\Delta$

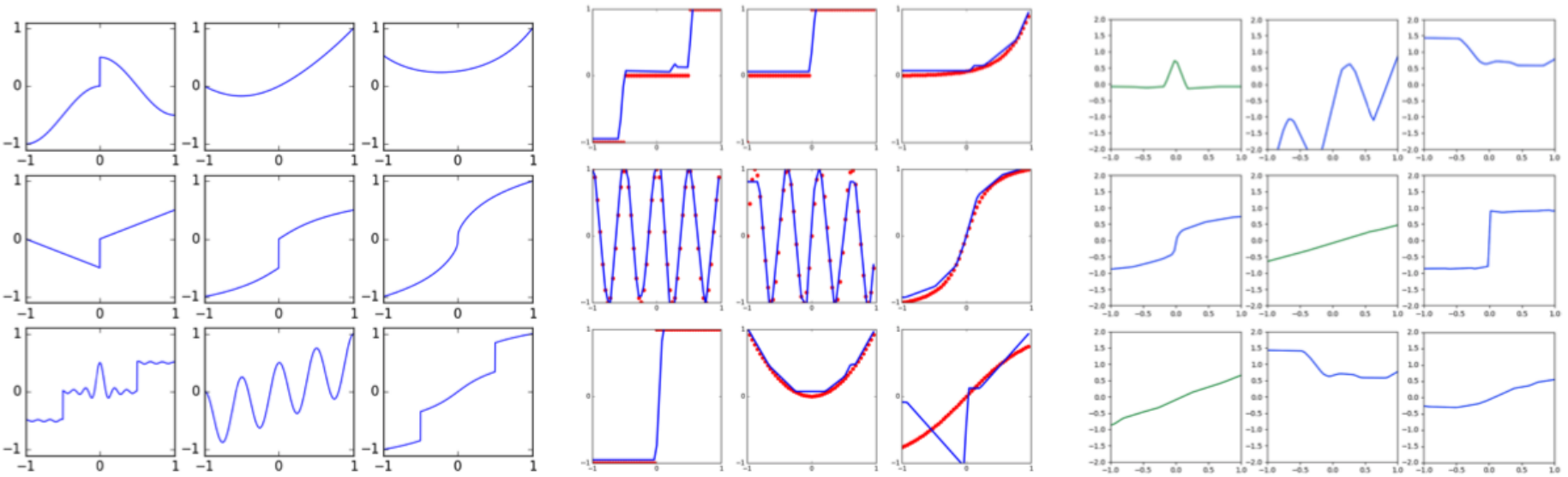


Figure 3: Random functions (left); BOUNCEGRAD (center) and MOMA (right) modules. All but one BOUNCEGRAD modules(blue) are nearly identical to a basis function(red).

meta relational learning

Modules to be learned: node embedding m_{s_i} and edge embedding $m_{s_{ij}}$.
Models: message passing.

$$s_i^{t+1} = s_i^t + m_{S_i}\left(s_i^t, \sum_{j \in neigh(v_i)} \mu_{ji}^t\right)$$

where $\mu_{ji}^t = m_{S_{ij}}(s_i^t, s_j^t)$
Applying this procedure T times to get $s^{t+1}, \cdots, s^T$; the whole process is differentiable, allowing us to train the parameters of $m_{S_i}, m_{S_{ij}}$ end-to-end based on predictive loss.

Algorithm 1 `BounceGrad` with learned proposal function for Graph Neural Networks.

procedure INITIALIZESTRUCTURE($\mathcal{G}, \mathbb{G}, \mathbb{H}$) ▷ Initialize with random modules
  for $n_i \in \mathcal{G}.nodes$ **do** $S.m_{n_i} \leftarrow$random_elt($\mathbb{G}$)
  for $e_i \in \mathcal{G}.edges$ **do** $S.m_{e_i} \leftarrow$random_elt($\mathbb{H}$)
procedure PROPOSECANDIDATESTRUCTURE($S, \mathcal{G}, \mathbb{G}, \mathbb{H}, p$)
  $C \leftarrow S$
  $idx \leftarrow random\_elt(\mathcal{G}.nodes)$
  if Bernouilli$(1/2)$ **then** ▷ In our experiments we only have one node module and skip this branch
    $C.m_{n_{idx}} \leftarrow$random_elt($\mathbb{G} \setminus C.m_{n_{idx}}, p_{n_{idx}}$)
  else ▷ Resample incoming edges to one particular node
    for $e \in$ incoming($\mathcal{G}.nodes_{idx}$) **do**
      $C.m_e \leftarrow$random_elt($\mathbb{H}, p_{m_e}$)
  return P
procedure EVALUATE($\mathcal{G}, S, \mathcal{L}, \boldsymbol{x}, \boldsymbol{y}$)
  $\boldsymbol{w} \leftarrow \text{MP}^{(T)}(\mathcal{G}, S)(\boldsymbol{x})$ ▷ Running the GNN with modular structure S
  return $\mathcal{L}(\boldsymbol{y}, \boldsymbol{w})$
procedure BOUNCEGRAD($\mathcal{G}, \mathbb{G}, \mathbb{H}, \mathcal{T}^1, \ldots, \mathcal{T}^k$) ▷ Modules in $\mathbb{G}, \mathbb{H}$ and proposal P start untrained
  for $l \in [1, k]$ **do**
    $S^l \leftarrow$ InitializeStructure($\mathcal{G}, \mathbb{G}, \mathbb{H}$)
  while not done **do**
    $l \leftarrow random\_elt([1, k])$
    $(\boldsymbol{x}, \boldsymbol{y}) \leftarrow$ sample($\mathcal{T}^l$) ▷ Train data
    $p \leftarrow P(\boldsymbol{x}, \boldsymbol{y})$ ▷ Proposal function predicts probabilities for every module slot
    $C \leftarrow$ ProposeCandidateStructure($S^l, \mathcal{G}, \mathbb{G}, \mathbb{H}, p$)
    $L_{S^l} \leftarrow$Evaluate($\mathcal{G}, S^l, \mathcal{L}, \boldsymbol{x}, \boldsymbol{y}$)
    $L_C \leftarrow$Evaluate($\mathcal{G}, C, \mathcal{L}, \boldsymbol{x}, \boldsymbol{y}$)
    $S^l \leftarrow$SimulatedAnnealing($(S^l, L_{S^l}), (C, L_C)$) ▷ Choose between S^l and C
    $(\boldsymbol{x}', \boldsymbol{y}') \leftarrow$ sample($\mathcal{T}^l$) ▷ Test data
    $L, L_P \leftarrow$Evaluate($\mathcal{G}, S^l, \mathcal{L}, \boldsymbol{x}', \boldsymbol{y}'), \mathcal{L}_P(p, S)$
    for $h \in \mathbb{H}$ **do** $\theta_h \leftarrow$ GradientDescent(L, θ_h)
    for $g \in \mathbb{G}$ **do** $\theta_g \leftarrow$ GradientDescent(L, θ_g)
    $\theta_P \leftarrow$ GradientDescent(L_P, θ_p)
  return $\mathbb{G}, \mathbb{H}, P$ ▷ Return specialized modules and proposal function

- S: a structure, one module per node $m_{n_1}, \ldots, m_{n_r}$, one module per edge $m_{e_1}, \ldots, m_{e_{r'}}$. Each m_{n_i} is a pointer to $\mathbb{G}$ and each m_{e_j} is a pointer to $\mathbb{H}$.
- $\mathcal{T}^1, \ldots, \mathcal{T}^k$: set of regression tasks, from which we can sample (x, y) pairs.
- $\text{MP}^{(T)}(\mathcal{G}, S)(x_t) \to x_{t+1}$: message-passing function applied T times, see Gilmer et al. (2017) for details.
- $\mathcal{L}(y_{target}, y_{pred})$: loss function; in our case $|y_{target} - y_{pred}|^2$.
- P proposal function, a neural network that returns a factored probability distribution, with the probability for each module for each node and each edge.
- $\mathcal{L}_P(p, S)$: loss function for proposal function; in our case the cross-entropy loss function of probability p to predict S.
- L: instantiations losses. This includes the actual loss value and infrastructure to backpropagate them.
- random_elt($\mathbb{S}$, P): pick element from set $\mathbb{S}$ according to a probability distribution

Automated relational meta learning

Huaxiu Yao, Xian Wu, Zhiqiang Tao, Yaliang Li, Bolin Ding, Ruirui Li, Zhenhui Li

ICLR 2020

Meta learning category:

- black-box based meta learner: L2L by gradient descent by gradient descent.
- optimization based meta learner: MAML type.
- metric based meta learner: Siamese neural network.

Basic assumption:
Tasks are related, specifically, $\mathcal{T}_i \sim P(\mathcal{T})$

Limitations:

globally shared meta learners fail to handle tasks from different distributions, which is known as task heterogeneity.

Related works:
Customize globally shared meta-learner for each task:

- Probabilistic meta learning
- Hierarchically structured meta learning

Main idea:

- a meta-knowledge graph, which is designed to organize and memorize historical learned knowledge
- a task specific prototype-based graph, which taps into the meta-knowledge graph to acquire relevant knowledge for enhancing its own representation
- utilize the enhanced prototype-based representation to customize the shared meta learner.

Task $\mathcal{T}_i$ specific prototype-based relational graph:

Vertex:

$$c_i^k = \frac{1}{N_k^{tr}} \sum_{j=1}^{N_k^{tr}} \epsilon(x_j),$$

where N_k^{tr} denotes the number of samples in class k

Relational graph:

$$A_{R_i}(c_i^j, c_i^m) = \sigma(W_r(|c_j^i - c_m^i|/\gamma_r) + b_r).$$

Global meta-knowledge graph:

$G = (H_G, A_G)$, where $H_G = \{h_j | \forall j \in [1, G]\} \in R^{G \times d}$ and $A_G = \{A_G(h_j, h_m) | j, m \in [1, G]\} \in R^{G \times G}$

Weight:

$$A_G(h^j, h^m) = \sigma(W_o(|h^j - h^m|/\gamma_o) + b_o).$$

To enhance the learning of new tasks with involvement of historical knowledge.

Construct a super-graph:$S_i = (A_i, H_i)$, where

$$A_i = (A_{R_i}, A_S; A_S^T, A_G) \in R^{(K+G)\times(K+G)}$$

$$H_i = (C_{R_i}; H_G) \in R^{(K+G)\times d}$$

$$A_S(c_i^j, h^k) = \frac{\exp(-||(c_i^j - h^k)/\gamma_s||^2/2)}{\sum_{k'=1}^{K} \exp(-||(c_i^j - h^{k\prime})/\gamma_s||^2/2)}$$

Propagate knowledge from meta-knowledge graph G to the prototype-based relational graph R_i by GNN.

$$H^{(l+1)} = MP(A_i, H^{(l)}; W^{(l)}),$$

Information-propagated feature representation for the prototype-based relational graph R_i as the top-K rows of $H^{(L)}$, which is denoted as $\hat{C}_{R_i} = \{\hat{c}_i^j | j \in [1, K]\}$.

Task-specific knowledge fusion and adaptation.

Two vertex sets C_{R_i} (the raw prototype graph) and $\hat{C}_{R_i}$ (prototype representations after absorbing the relevant knowledge from the meta-knowledge graph) contribute the most to the creation of the task-specific meta-learner.

To get the dense representation:

$$q_i = MeanPool(AG^q(C_{R_i})) = mean(AG^q(c_i^j)), Lq = ||C_{R_i} - AG^q_{dec}(AG^q(C_{R_i}))$$

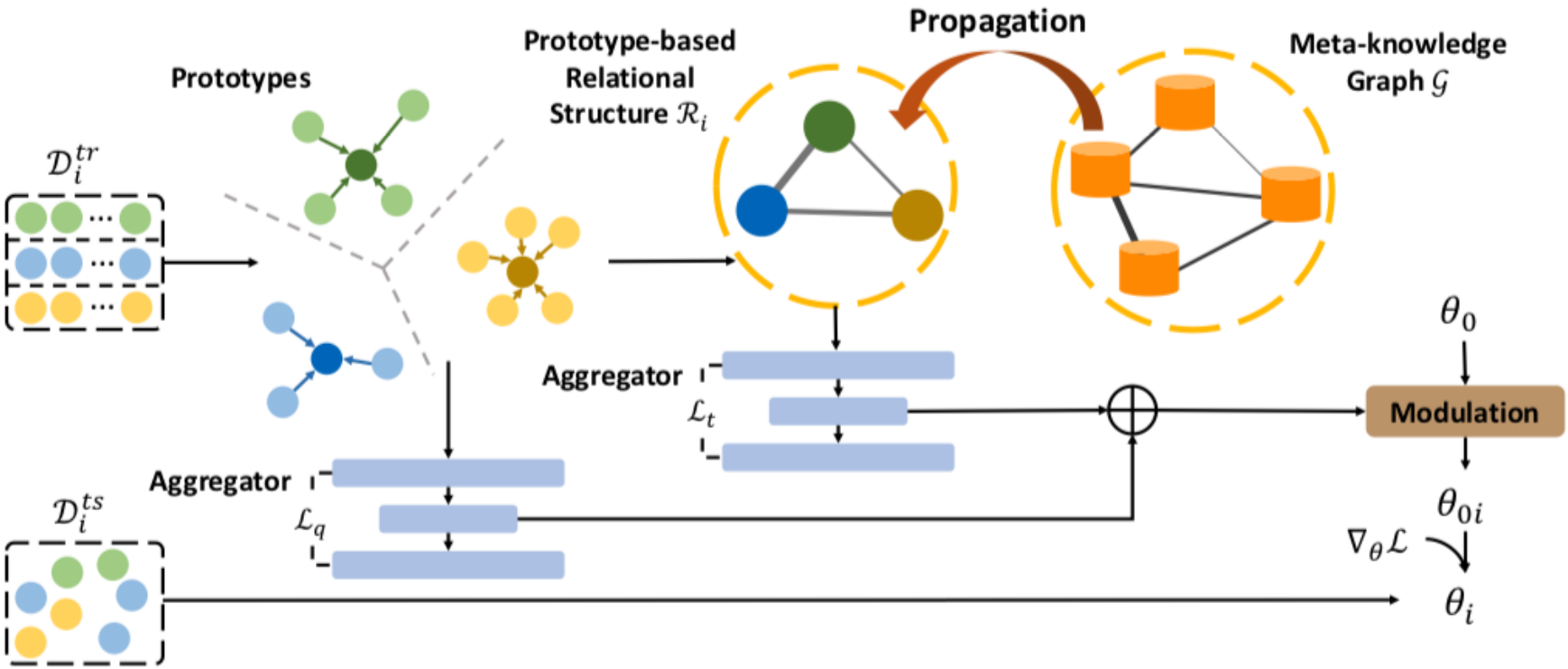
$$t_i = MeanPool(AG^q(\hat{C}_{R_i})) = mean(AG^q(\hat{c}_i^j)), Lt = ||\hat{C}_{R_i} - AG^q_{dec}(AG^q(\hat{C}_{R_i}))|$$

tailor the task-specific information to the globally shared initialization θ_0:

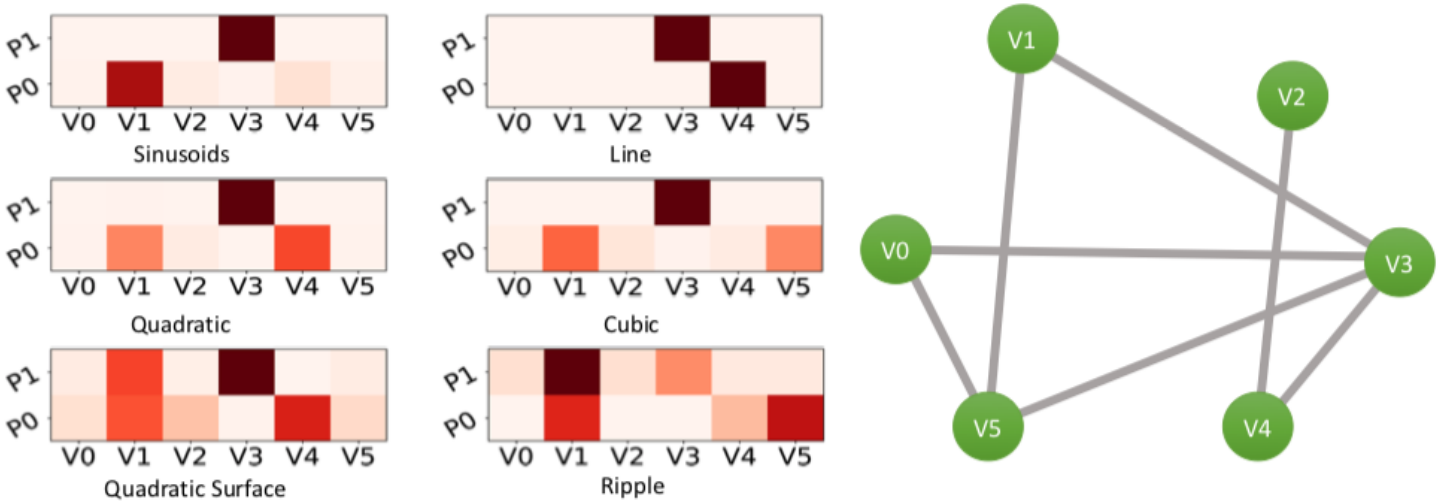
$$\theta_{0i} = \sigma(W_g(ti \oplus qi) + bg) \circ \theta_0,$$

Loss function:

$$\sum_{i=1}^{I} L(f_{\theta_{0i} - \alpha \nabla L(f_\theta, D_i^{tr})}, D_i^{ts}) + \mu_1 L_t + \mu_2 L_q,$$


Prototypes
Prototype-based Relational Structure $\mathcal{R}_i$
Propagation
Meta-knowledge Graph $\mathcal{G}$
$\mathcal{D}_i^{tr}$
Aggregator
$\mathcal{L}_t$
θ_0
Modulation
Aggregator
$\mathcal{L}_q$
$\mathcal{D}_i^{ts}$
θ_{0i}
$\nabla_\theta \mathcal{L}$
θ_i

(1) Sinusoids: $z(x, y) = assin(wsx + bs)$, where
$as \sim U[0.1, 5.0], bs \sim U[0, 2] ws \sim U[0.8, 1.2]$;
(2) Line: $z(x, y) = alx + bl$, where $al \sim U[3.0, 3.0], bl \sim U[3.0, 3.0]$;
(3) Quadratic: $z(x, y) = aqx2 + bqx + cq$, where
$aq \sim U[0.2, 0.2], bq \sim U[2.0, 2.0], cq \sim U[3.0, 3.0]$;
(4) Cubic: $z(x, y) = acx3 + bcx2 + ccx + dc$, where
$ac \sim U[0.1, 0.1], bc \sim U[0.2, 0.2], cc \sim U[2.0, 2.0], dc \sim U[3.0, 3.0]$;
(5) Quadratic Surface: $z(x, y) = aqsx2 + bqsy2$, where
$aqs \sim U[1.0, 1.0], bqs \sim U[1.0, 1.0]$;
(6) Ripple: $z(x, y) = sin(ar(x2 + y2)) + br$, where
$ar \sim U[0.2, 0.2], br \sim U[3.0, 3.0]$.



Model	MAML	Meta-SGD	BMAML	MT-Net	MUMOMAML	HSML	ARML
10-shot	2.29 ± 0.16	2.91 ± 0.23	1.65 ± 0.10	1.76 ± 0.12	0.52 ± 0.04	0.49 ± 0.04	$\mathbf{0.44 \pm 0.03}$

Figure 2: In the top figure, we show the interpretation of meta-knowledge graph. The left heatmap shows the similarity between prototypes (P0, P1) and meta-knowledge vertices (V0-V5). The right part show the meta-knowledge graph. In the bottom table, we show the overall performance (mean square error with 95% confidence) of 10-shot 2D regression.