Learning to Propagate for Graph Meta-Learning

Lu Liu[1] *Tianyi Zhou*[2] *Guodong Long*[1] *Jing Jiang*[1] *Chengqi Zhang*[1]

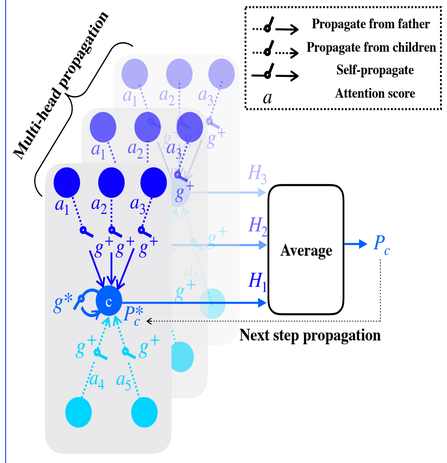
[1]University *of* Technology Sydney

[2]University *of* Washington

INTRODUCTION

We show that a **meta-learner that explicitly relates tasks on a graph** describing the relations of their output dimensions (e.g., classes) can significantly improve the performance of few-shot learning. This type of graph is usually **free or cheap** to obtain but has rarely been explored in previous works. We study the **prototype based few-shot classification**, in which a prototype is generated for each class, such that the nearest neighbor search between the prototypes produces an accurate classification. We introduce **``Gated Propagation Network (GPN)",** which **learns to propagate messages between prototypes of different classes on the graph**, so that learning the prototype of each class benefits from the data of other related classes. In GPN, an attention mechanism is used for the aggregation of messages from neighboring classes, and a **gate** is deployed to choose between the aggregated messages and the message from the class itself. **GPN is trained on a sequence of tasks from many-shot to few-shot generated by subgraph sampling**. During training, it is able to **reuse and update previously achieved prototypes** from the memory in a life-long learning cycle. In experiments, we change the training-test discrepancy and test task generation settings for thorough evaluations. GPN outperforms recent meta-learning methods on two benchmark datasets in all studied cases.

MODEL: GATED PROPAGATION NETWORK



Prototype propagation in GPN:
in each step t+1, each class y aggregates prototypes from its neighbors (parents and children) by multi-head attention, and chooses between the aggregated message or the message from itself by a gate g.

An **initial prototype** for each class y by averaging over all the K-shot samples belonging to class y as in prototypical networks:

$$\boldsymbol{P}_y^0 \triangleq \frac{1}{|\{(\boldsymbol{x}_i, y_i) \in \mathcal{D}^T : y_i = y\}|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}^T, y_i = y} f(\boldsymbol{x}_i).$$

At step t, for each class y, we firstly compute the **aggregated messages from its neighbors** N_y by a dot-product attention module a(p, q), i.e.,

$$\boldsymbol{P}_{\mathcal{N}_y \to y}^{t+1} \triangleq \sum_{z \in \mathcal{N}_y} a(\boldsymbol{P}_y^t, \boldsymbol{P}_z^t) \times \boldsymbol{P}_z^t, \quad a(p,q) = \frac{\langle h_1(p), h_2(q) \rangle}{\|h_1(p)\| \times \|h_2(q)\|}.$$

Then we apply a **gate** g to make decisions of whether accepting messages from its neighbors $\boldsymbol{P}_{\mathcal{N}_y \to y}^{t+1}$ or message from itself $\boldsymbol{P}_{y \to y}^{t+1}$, i.e.

$$\boldsymbol{P}_y^{t+1} \triangleq g\boldsymbol{P}_{y \to y}^{t+1} + (1-g)\boldsymbol{P}_{\mathcal{N}_y \to y}^{t+1}, \quad g = \frac{\exp[\gamma \cos(\boldsymbol{P}_y^0, \boldsymbol{P}_{y \to y}^{t+1})]}{\exp[\gamma \cos(\boldsymbol{P}_y^0, \boldsymbol{P}_{y \to y}^{t+1})] + \exp[\gamma \cos(\boldsymbol{P}_y^0, \boldsymbol{P}_{\mathcal{N}_y \to y}^{t+1})]},$$

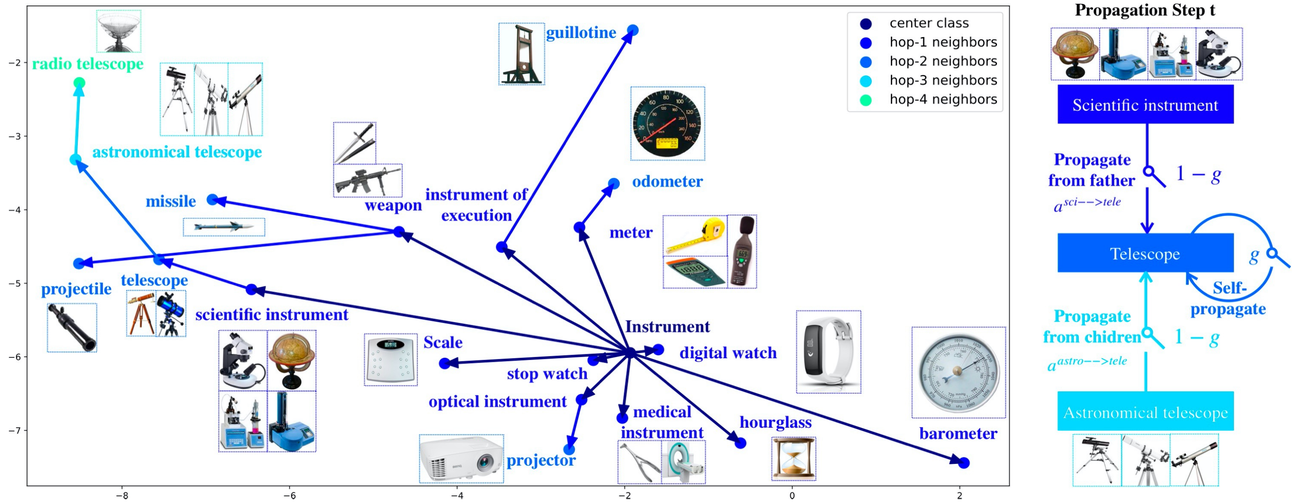
To capture different types of relation and jointly use them for propagation, we aggregate k attentive and gated propagation modules with untied parameters

$$\boldsymbol{P}_y^{t+1} = \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{P}_y^{t+1}[i]$$

The **final prototype** is given as the weighted sum of the initial prototype and the refined prototype:

$$\boldsymbol{P}_y \triangleq \lambda \times \boldsymbol{P}_y^0 + (1-\lambda) \times \boldsymbol{P}_y^{\mathcal{T}}$$

OVERVIEW : AN INTUITIVE UNDERSTANDING OF GRAPH META-LEARNING



LEFT: Visualization of the class prototypes produced by GPN for few-shot tasks and the associated graph. **RIGHT**: GPN's propagation mechanism for one step: for each node, its neighbors pass messages (their prototypes) to it according to attention weight a, where a gate further choose to accept the message from the neighbors g+ or from the class itself g*.

TRAINING STRATEGIES

Generating training tasks by subgraph sampling:
random sampling and snowball sampling.
Random sampling captures strongly-related classes
Snowball sampling captures weakly-related classes

Building propagation pathways by training on maximum spanning trees
Only propagate through the most related / close classes according to cosine similarity

Curriculum learning
Early stage: traditional supervised learning tasks
Later stage: train on few-shot learning tasks

EXPERIMENTAL RESULTS

Validation accuracy (mean±CI%95) on 600 test tasks achieved by GPN and baselines on *tiered*ImageNet-Close with few-shot tasks generated by random sampling.

Model	5way1shot	5way5shot	10way1shot	10way5shot
Prototypical Net [23]	42.87±1.67%	62.68±0.99%	30.65±1.15%	48.64±0.70%
GNN [6]	42.33±0.80%	59.17±0.69%	30.50±0.57%	44.33±0.72%
Closer Look [3]	35.07±1.53%	47.48±0.87%	21.58±0.96%	28.01±0.40%
PPN [15]	41.60±1.59%	63.04±0.97%	28.48±1.09%	48.66±0.70%
GPN	48.37±1.80%	64.14±1.00%	33.23±1.05%	50.50±0.70%
GPN+	**50.54±1.67%**	**65.74±0.98%**	**34.74±1.05%**	**51.50±0.70%**

See our paper for more results on *tiered*ImageNet-Close and *tiered*ImageNet-Far by different subgraph sampling strategies: random sampling and snowball sampling.
We found 1) propagation is more effective between close classes
2) propagation improves the performance both when discriminating between close classes (snowball sampling) and far classes (random sampling)