Position-aware Graph Neural Networks

[ICML 2019 oral]

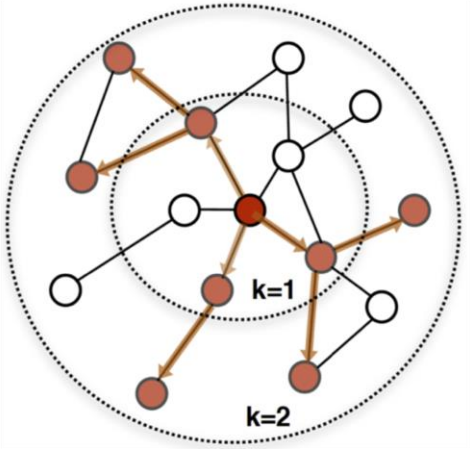
Seungryong Yoo

Seoul National University

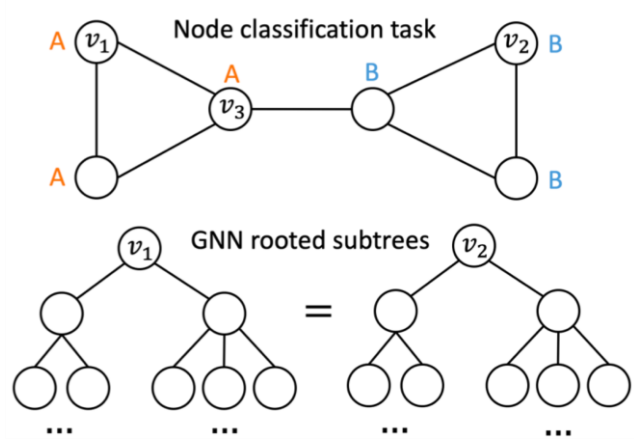
Position-aware Graph Neural Networks

Motivation

- **Current GNN-based node embedding methods only focus on local neighborhood structure**
  (e.g. q - h o p neighbor)
- **Without node features, GNN cannot distinguish between two nodes at different location, but having the same local structure**



from [GraphSAGE 2017]



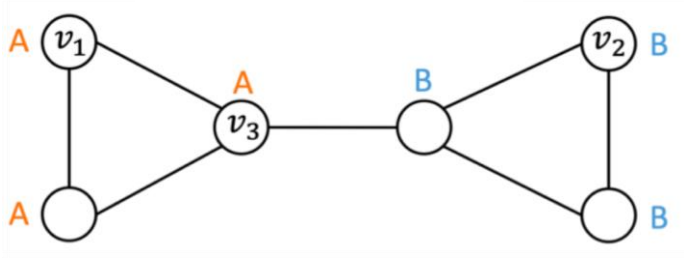
Position-aware Graph Neural Networks

Why node position information is important?

- How can we decide node position?
  → compare distance from common reference

- d_{sp} : shortest path distance between two nodes
  $d_{sp}(v_1, v_3) = 1$, $d_{sp}(v_2, v_3) = 2$
  → now v_1 and v_2 are distinguishable

- call these reference as anchor-set



Position-aware Graph Neural Networks

Definition of structure-aware/position-aware node embedding

- **Position-aware node embedding**

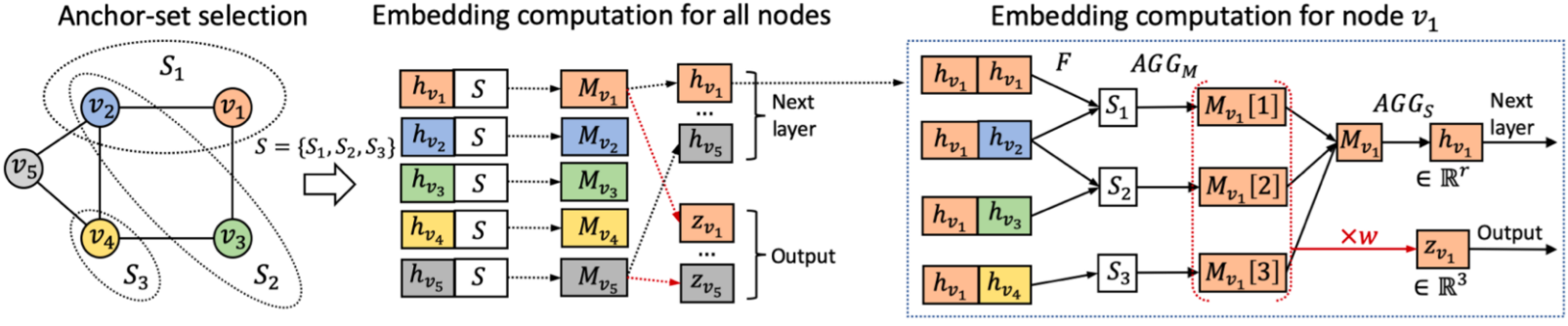
Definition 1. *A node embedding* $\mathbf{z}_i = f_p(v_i), \forall v_i \in \mathcal{V}$ *is position-aware if there exists a function* $g_p(\cdot,\cdot)$ *such that* $d_{sp}(v_i, v_j) = g_p(\mathbf{z}_i, \mathbf{z}_j)$*, where* $d_{sp}(\cdot,\cdot)$ *is the shortest path distance in* G*.*

- **Structure-aware node embedding**

Definition 2. *A node embedding* $\mathbf{z}_i = f_{s_q}(v_i), \forall v_i \in \mathcal{V}$ *is structure-aware if it is a function of up to* q*-hop network neighbourhood of node* v_i*. Specifically,* $\mathbf{z}_i = g_s(N_1(v_i), ..., N_q(v_i))$*, where* $N_k(v_i)$ *is the set of the nodes* k*-hops away from node* v_i*, and* g_s *can be any function.*

Position-aware Graph Neural Networks

Method Overview

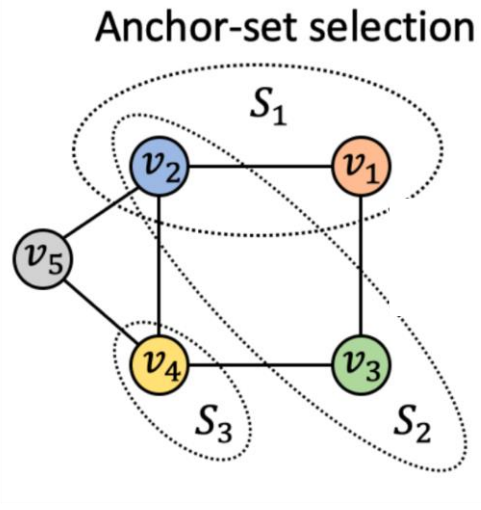


S_i : *i-th* anchor-set

S : set of anchor-sets

Position-aware Graph Neural Networks

Method Overview : Anchor-set Selection



- **randomly select k anchor-sets for each forward pass(i.e. each layer in network)** (sampling guided by *Bourgain Theorem*)

Position-aware Graph Neural Networks

Method Overview : Anchor-set Selection

- **Bourgain Theorem**

  → It is a guidance for choice of anchor-sets to guarantee the resulting representations to have low-distortion

- **What is low-distortion?**

Definition 3. *Given two metric spaces* $(\mathcal{V}, d)$ *and* $(\mathcal{Z}, d')$ *and a function* $f : \mathcal{V} \to \mathcal{Z}$, f *is said to have distortion* α *if* $\forall u, v \in \mathcal{V}$, $\frac{1}{\alpha} d(u, v) \leq d'(f(u), f(v)) \leq d(u, v)$.

→ a low-distortion embedding function preserves distance well when mapping from one metric space to another metric space

Position-aware Graph Neural Networks

Method Overview : Anchor-set Selection

- **Bourgain Theorem**

Theorem 1. *(Bourgain theorem) Given any finite metric space $(\mathcal{V}, d)$ with $|\mathcal{V}| = n$, there exists an embedding of $(\mathcal{V}, d)$ into $\mathbb{R}^k$ under any l_p metric, where $k = O(\log^2 n)$, and the distortion of the embedding is $O(\log n)$.*

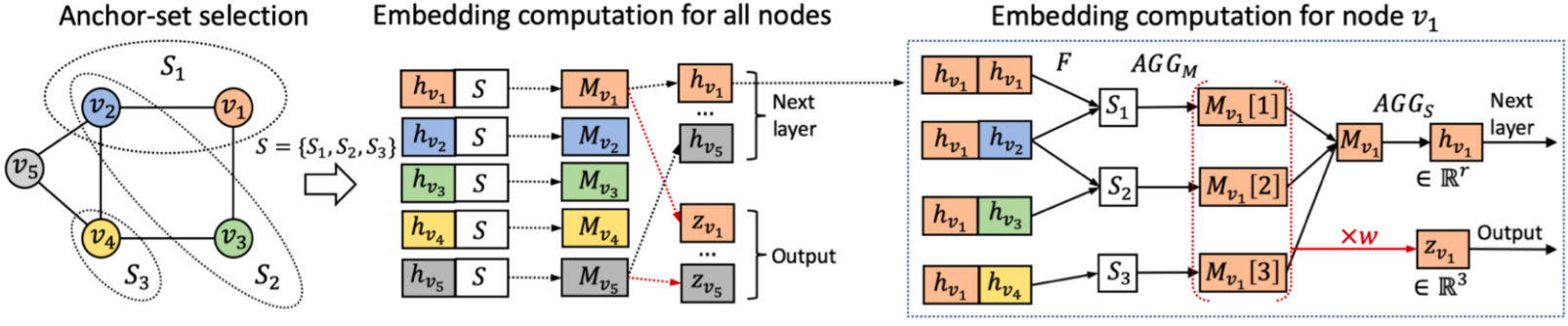
Theorem 2. *(Constructive proof of Bourgain theorem) For metric space $(\mathcal{V}, d)$, given $k = c\log^2 n$ random sets $S_{i,j} \subset \mathcal{V}, i = 1, 2, ..., \log n, j = 1, 2, ..., c\log n$ where c is a constant, $S_{i,j}$ is chosen by including each point in $\mathcal{V}$ independently with probability $\frac{1}{2^i}$. An embedding method for $v \in \mathcal{V}$ is defined as:*

$$f(v) = \left(\frac{d(v, S_{1,1})}{k}, \frac{d(v, S_{1,2})}{k}, ..., \frac{d(v, S_{\log n, c\log n})}{k}\right) \quad (1)$$

where $d(v, S_{i,j}) = \min_{u \in S_{i,j}} d(v, u)$. Then, f is an embedding method that satisfies Theorem 1.

Position-aware Graph Neural Networks

Method Overview



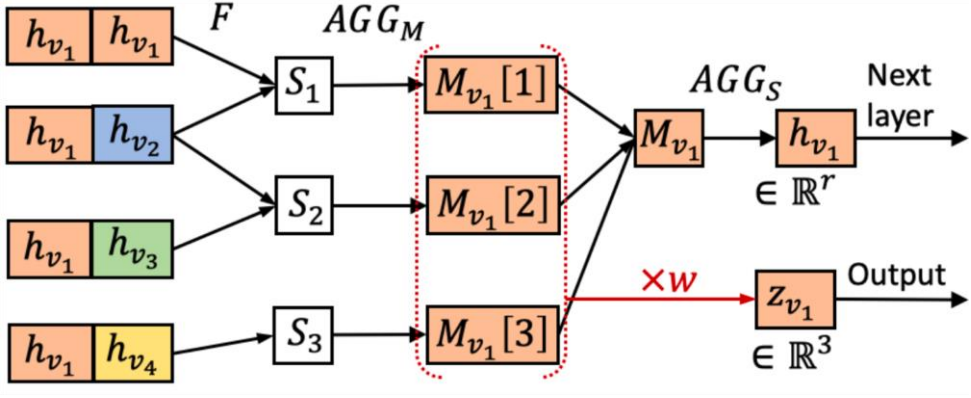
$\boldsymbol{h_{v_i}}$: node feature from *(l-1)-th* layer

$\boldsymbol{F}$: message computation function between two nodes

$\boldsymbol{AGG_M}$: message aggregation function within each anchor-set

Position-aware Graph Neural Networks

Method Overview : Message Computation

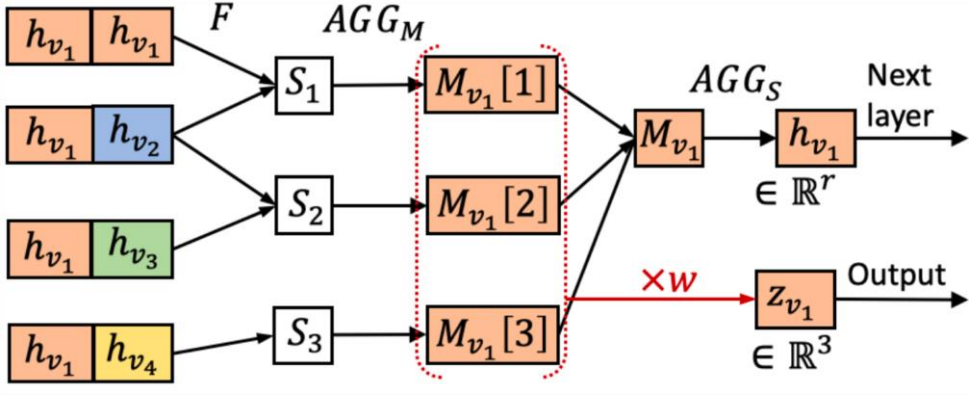


$$d_{sp}^{q}(u,v) = \begin{cases} d_{sp}(u,v), & if\ d_{sp}(u,v) \le q \\ \infty, & otherwise \end{cases}$$

- compute messages between query node and each node in each anchor-set (considering both position similarity and node features)

Position-aware Graph Neural Networks

Method Overview : Message Computation



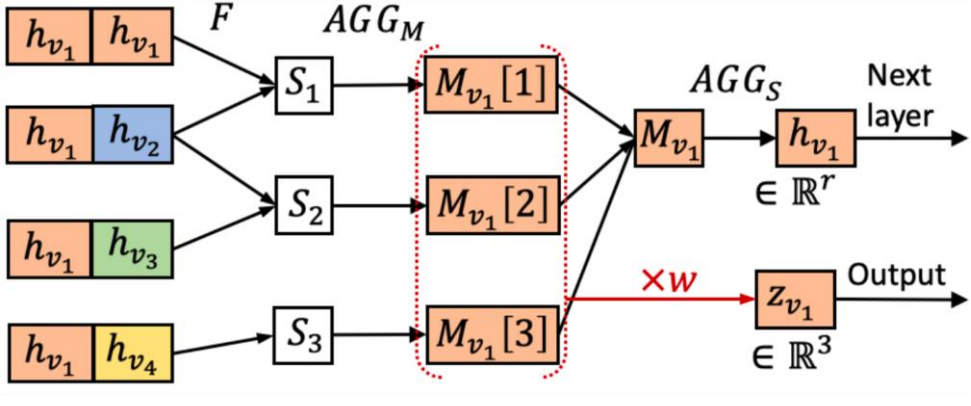
$$F\left(v_i, v_j, h_{v_i}, h_{v_j}\right) = s(v_i, v_j)\,CONCAT\left(h_{v_i}, h_{v_j}\right)$$

$$s(v_i, v_j) = \frac{1}{d_{sp}^{q}(v_i, v_j) + 1}$$

- compute messages between query node and each node in each anchor-set (considering both position similarity and node features)

Position-aware Graph Neural Networks

Method Overview : Message Aggregation



$$for\ anchor\ set\ S_t$$

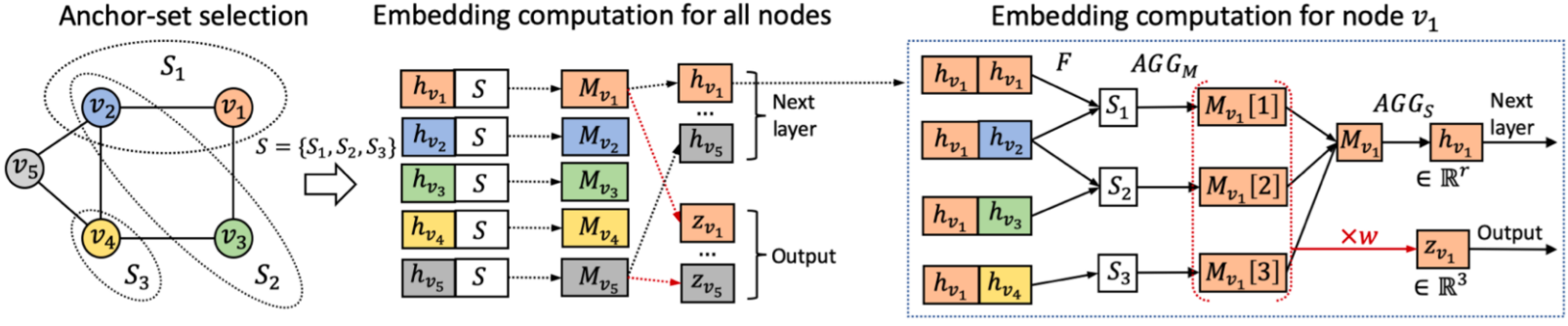
$$M_{v_i}[t] = AGG_M\left(\left\{F\left(v_i, v_j, h_{v_i}, h_{v_j}\right), v_j \in S_t\right\}\right)$$

$$M_{v_i} \in R^{k \times m}$$

- aggregate messages within each anchor-set
  → output a matrix, in which each row is the information from each anchor-set

Position-aware Graph Neural Networks

Method Overview

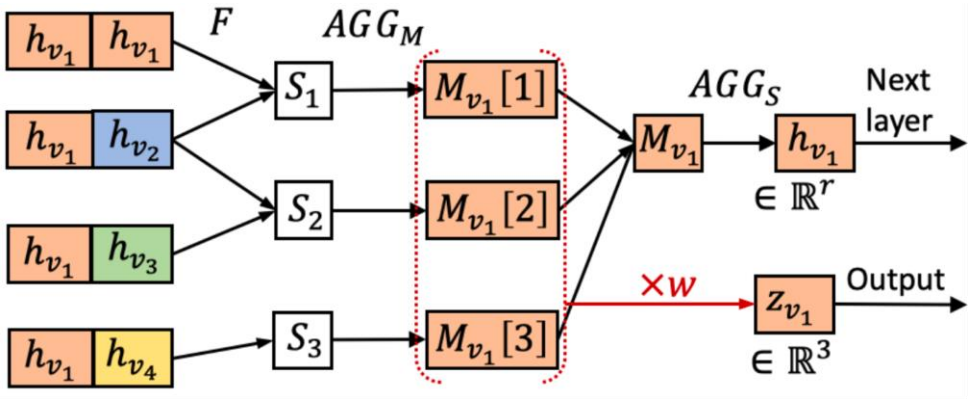


AGG_S : message aggregation function across all anchor-sets

w : vector which projects $\mathbb{R}^r$ to low-dimension vector

Position-aware Graph Neural Networks

Method Overview : Message Computation



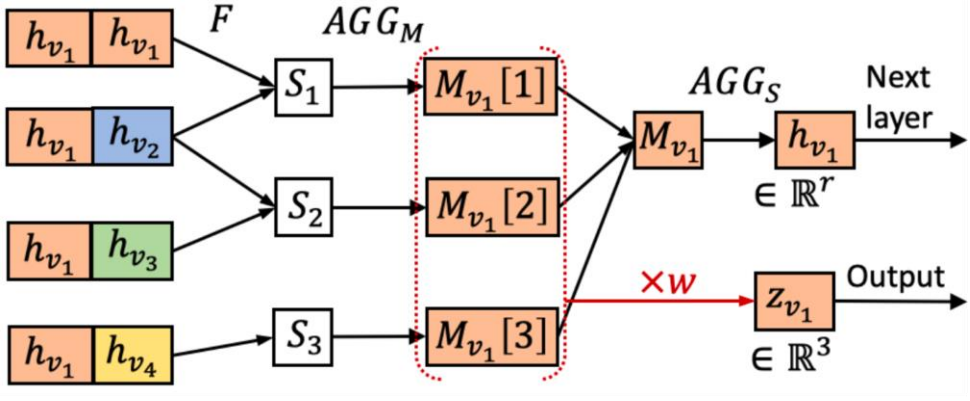
$$w \in R^{m \times 1}$$

$$z_{v_i} = \sigma(M_{v_i} w) \in R^{k \times 1}$$

- project output matrix to low-dimension(=number of anchor-sets) vector
- each element of the low-dimension vector encodes the distance information for each anchor-sets, therefore structurally equivalent nodes are distinguishable

Position-aware Graph Neural Networks

Method Overview : Message Computation



$$h_{v_i} = AGG_S(M_{v_i})$$

- aggregate messages across all the anchor-sets
  → computed new node feature passed to next layer

Position-aware Graph Neural Networks

Proposed Algorithm

Algorithm 1 The framework of P-GNNs

Input: Graph $G = (\mathcal{V}, \mathcal{E})$; Set S of k anchor-sets $\{S_i\}$; Node input features $\{\mathbf{x}_v\}$; Message computation function F that outputs an r dimensional message; Message aggregation functions $\text{AGG}_M, \text{AGG}_S$; Trainable weight vector $\mathbf{w} \in \mathbb{R}^r$; Non-linearity σ; Layer $l \in [1, L]$

Output: Position-aware embedding $\mathbf{z}_v$ for every node v

$\mathbf{h}_v \leftarrow \mathbf{x}_v$

for $l = 1, \ldots, L$ **do**

  $S_i \sim \mathcal{V}$ for $i = 1, \ldots, k$ → anchor-set selection

  for $v \in \mathcal{V}$ **do**

    $\mathbf{M}_v = \mathbf{0} \in \mathbb{R}^{k \times r}$

    for $i = 1 \ldots, k$ **do**

      $\mathcal{M}_i \leftarrow \{F(v, u, \mathbf{h}_v, \mathbf{h}_u), \forall u \in S_i\}$ → message computation between nodes

      $\mathbf{M}_v[i] \leftarrow \text{AGG}_M(\mathcal{M}_i)$ → message aggregation within anchor-set

    end for

    $\mathbf{z}_v \leftarrow \sigma(\mathbf{M}_v \cdot \mathbf{w})$ → project to low-dimension distance vector

    $\mathbf{h}_v \leftarrow \text{AGG}_S(\{\mathbf{M}_v[i], \forall i \in [1, k]\})$ → mess age aggregation across all the anchor-sets

  end for

end for

$\mathbf{z}_v \in \mathbb{R}^k, \forall v \in \mathcal{V}$

Position-aware Graph Neural Networks

Experiments

Table 1. P-GNNs compared to GNNs on link prediction tasks, measured in ROC AUC. Grid-T and Communities-T refer to the transductive learning setting of Grid and Communities, where one-hot feature vectors are used as node attributes. Standard deviation errors are given.

	Grid-T	Communities-T	Grid	Communities	PPI
GCN	0.698 ± 0.051	0.981 ± 0.004	0.456 ± 0.037	0.512 ± 0.008	0.769 ± 0.002
GraphSAGE	0.682 ± 0.050	0.978 ± 0.003	0.532 ± 0.050	0.516 ± 0.010	0.803 ± 0.005
GAT	0.704 ± 0.050	0.980 ± 0.005	0.566 ± 0.052	0.618 ± 0.025	0.783 ± 0.004
GIN	0.732 ± 0.050	0.984 ± 0.005	0.499 ± 0.054	0.692 ± 0.049	0.782 ± 0.010
P-GNN-F-1L	0.542 ± 0.057	0.930 ± 0.093	0.619 ± 0.080	0.939 ± 0.083	0.719 ± 0.027
P-GNN-F-2L	0.637 ± 0.078	$\mathbf{0.989} \pm 0.003$	0.694 ± 0.066	$\mathbf{0.991} \pm 0.003$	0.805 ± 0.003
P-GNN-E-1L	0.665 ± 0.033	0.966 ± 0.013	0.879 ± 0.039	0.985 ± 0.005	0.775 ± 0.029
P-GNN-E-2L	$\mathbf{0.834} \pm 0.099$	0.988 ± 0.003	$\mathbf{0.940} \pm 0.027$	0.985 ± 0.008	$\mathbf{0.808} \pm 0.003$

Position-aware Graph Neural Networks

Experiments

Table 2. Performance on pairwise node classification tasks, measured in ROC AUC. Standard deviation errors are given.

	Communities	Email	Protein
GAT	0.520 ± 0.025	0.515 ± 0.019	0.515 ± 0.002
GraphSAGE	0.514 ± 0.028	0.511 ± 0.016	0.520 ± 0.003
GAT	0.620 ± 0.022	0.502 ± 0.015	0.528 ± 0.011
GIN	0.620 ± 0.102	0.545 ± 0.012	0.523 ± 0.002
P-GNN-F-1L	0.985 ± 0.008	0.630 ± 0.019	0.510 ± 0.010
P-GNN-F-2L	0.997 ± 0.006	$\mathbf{0.640} \pm 0.037$	$\mathbf{0.729} \pm 0.176$
P-GNN-E-1L	0.991 ± 0.013	0.625 ± 0.058	0.507 ± 0.006
P-GNN-E-2L	$\mathbf{1.0} \pm 0.001$	$\mathbf{0.640} \pm 0.029$	0.631 ± 0.175

Position-aware Graph Neural Networks

Paper Reproducing

- **Author's original implementation**

  https://github.com/JiaxuanYou/P-GNN

- **Task**

  Link prediction

- **Used Dataset**

  Grid, Communities, PPI

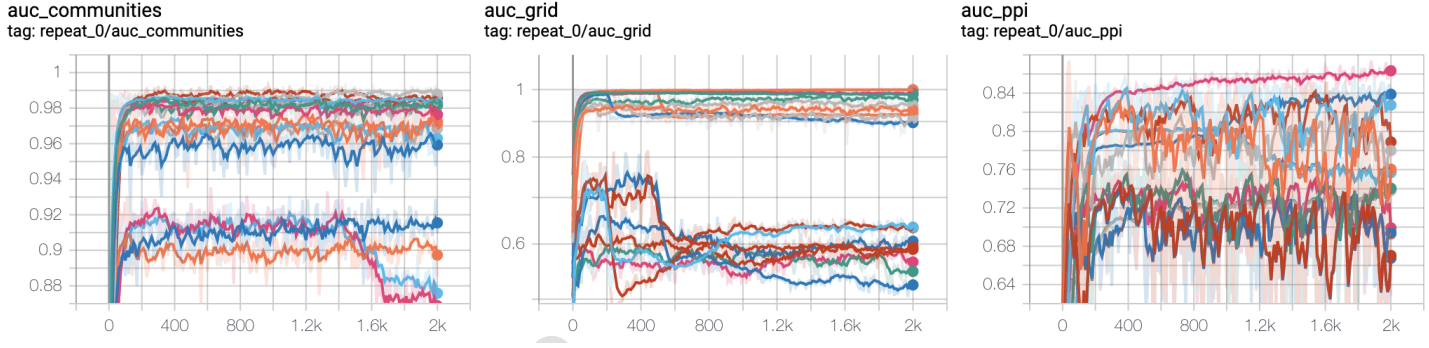
- **Experimental Setup**

P-GNN-E-1L : 1-layer P-GNN using exact shortest path distance
P-GNN-E-2L : 2-layer P-GNN using exact shortest path distance
P-GNN-E-3L : 3-layer P-GNN using exact shortest path distance
P-GNN-F-1L : 1-layer P-GNN using truncated 2-hop shortest path distance
P-GNN-F-2L : 2-layer P-GNN using truncated 2-hop shortest path distance
P-GNN-F-3L : 3-layer P-GNN using truncated 2-hop shortest path distance

Position-aware Graph Neural Networks

Paper Reproducing

- **Training log**



Grid, Communities datasets are relatively smaller than the PPI dataset, it took a few couple of minutes to train, however, for training the PPI dataset, it took almost 7~8 hours to train.

Position-aware Graph Neural Networks

Paper Reproducing

- **Results on reproducing experiments**

Table 1: Reproduced results

	Grid	Communities	PPI
P-GNN-E-1L	0.644±0.009	0.955±0.008	0.773±0.032
P-GNN-E-2L	0.922±0.012	0.983±0.002	0.781±0.008
P-GNN-E-3L	**0.930**±0.039	0.981±0.001	0.778±0.019
P-GNN-F-1L	0.541±0.037	0.951±0.012	0.747±0.006
P-GNN-F-2L	0.751±0.010	0.968±0.005	0.775±0.001
P-GNN-F-3L	0.708±0.023	**0.969**±0.006	**0.796**±0.021

- **Results on original paper**

Table 2: Original results

	Grid	Communities	PPI
P-GNN-E-1L	0.879±0.039	0.985±0.005	0.775±0.029
P-GNN-E-2L	**0.940**±0.027	0.985±0.008	**0.808**±0.003
P-GNN-F-1L	0.619±0.080	0.939±0.083	0.719±0.027
P-GNN-F-2L	0.694±0.066	**0.991**±0.003	0.805±0.003

Position-aware Graph Neural Networks

Paper Reproducing

- **Results on reproducing experiments**

Table 1: Reproduced results

	Grid	Communities	PPI
P-GNN-E-1L	0.644±0.009	0.955±0.008	0.773±0.032
P-GNN-E-2L	0.922±0.012	0.983±0.002	0.781±0.008
P-GNN-E-3L	**0.930**±0.039	0.981±0.001	0.778±0.019
P-GNN-F-1L	0.541±0.037	0.951±0.012	0.747±0.006
P-GNN-F-2L	0.751±0.010	0.968±0.005	0.775±0.001
P-GNN-F-3L	0.708±0.023	**0.969**±0.006	**0.796**±0.021

Even though the original paper did not reported on the 3-layer P-GNNs, in my reproducing experiments on link prediction task, 3-layer P-GNNs works better than 2-layer P-GNNs. For Grid dataset, truncated 2-hop shortest path distance was better than the case of using exact shortest path distance, however, for Communities and PPI dataset it showed reversed results.

Position-aware Graph Neural Networks

Paper Reproducing

- **Experiment requirements**

  python == 3.6.10
  torch == 1.1.0
  networkX
  tensorboardX
  sklearn

- **conda Environment Setup**

```
conda create -n pgnn python=3.6
conda activate pgnn
conda install pytorch==1.1.0 torchvision -c pytorch
conda install networkX
conda install sklearn
python3 -m pip install tensorboardX
```

- **Run training code**

  Ex)

  python3 main.py --model PGNN --layer_num 2 --dataset grid --task=link --approximate=-1
  (In this case, we train 2-layer P-GNN, with exact shortest path distance, on the Grid Dataset)

- **Computing resource**

  4 GPUs (GeForce GTX TITAN)

Position-aware Graph Neural Networks

Paper Reproducing

- **Issues on installing torch-geometric**

```
Building wheels for collected packages: torch-sparse
  Building wheel for torch-sparse (setup.py) ... error
  ERROR: Command errored out with exit status 1:
```

torch-geometric have dependency on torch-sparse, torch-scatter, torch-cluster,
but it was quite challenging to install those frameworks, so that I used the original code except the codes requiring torch-geometric. Fortunately, the P-GNN model implementation does not depend on torch-geometric. Some functions for data loading and data pre-processing for graph data depends on torch-geometric, but it was not that difficult to deviate by using alternative libraries like networkX. Also, I added torch_geometric.data.Data source code in the dataset.py. Therefore I added editted version of original code to perform link prediction task on Grid, Communities, PPI dataset with this file.

Position-aware Graph Neural Networks

Paper Reproducing

- **Code explanation – model.py**

PGNN_layer : Compute the each node's distance to preselected anchor sets(from preselect_anchor in utils.py, the feeding data have dists_max, dists_argams attribute which indicates the shortest distance and the corresponding neighboring nodes). The distance is defined by passed arguments(exact shortest path distance of truncated 2-hop shortest distance). The messages from each anchor sets to each node is concatenated with the node feature, and then through linear_hidden, linear_out_position and defined Nonlinearity module, it finally return the position-aware vector.

P-GNN : it consists of a module sequence, Linear unit, and defined numbers(by parsed arguments of layer_num) of PGNN_layer unit. Finally it returns only one vector which is position-aware.

Position-aware Graph Neural Networks

Paper Reproducing

- **Code explanation – dataset.py**

get_tg_dataset : return the list of preprocessed graph data, masking for positive neighbor nodes, also it saves the distance information for each node to pickle file, and if it already exists use the saved caches to preprocess.

load_graphs : return the graph dataset from the saved data directory or from networkX. dataset to be returned is designated by its input argument dataset_str.

nx_to_tg_data : from load_graph function, we get graph data which is defined by networkX framework. So that this function preprocess this networkX graph data to be torch_geomtric.data.Data type.

load_tg_dataset : return torch_geometric.data.Data graph object dataset using its input argument dataset_name

Data : source code from torch_geometric.data.Data object (to deviate the problems triggered by uninstallation issue of torch_geometric and torch_sparse, torch_scatter, torch_cluster)

Position-aware Graph Neural Networks

Paper Reproducing

- **Code explanation – args.py**

--task : choose the task among link prediction or pairwise node classification
--model : choose model architecture among P-GNN, GCN, GAT, SAGE, GIN
--gpu : whether to use gpu
--cuda : which gpu to use
--dataset : which dataset to use, PPI, Grid, Communities
--approximate : -1 = exact shortest path distance, 2=truncated 2-hop shortest path distance
--layer_num : layer numbers of PGNN_layer for the PGNN
--cache : whether to use saved distance information of graph

Position-aware Graph Neural Networks

Paper Reproducing

- **Code explanation – main.py**

main : from the parsed arguments, it assigns the designated gpu, define the log writer(loss, auc which would be visualized using tensorboard)and load the dataset and model. And then during the defined repeat number and epoch(from parsed arguments), it computes the auc score and loss of train/val/test and then prints out to the console. Also, it saves the eventfile by the SummaryWriter. Finally, it saves the final score(mean and std) of auc in the results directory.

Thank you