24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

PushNet: Efficient and Adaptive Neural Message Passing

Julian Busch [1][2] and **Jiaxing Pi** [1] and **Thomas Seidl** [2]

Abstract. Message passing neural networks have recently evolved into a state-of-the-art approach to representation learning on graphs. Existing methods perform synchronous message passing along all edges in multiple subsequent rounds and consequently suffer from various shortcomings: Propagation schemes are inflexible since they are restricted to k-hop neighborhoods and insensitive to actual demands of information propagation. Further, long-range dependencies cannot be modeled adequately and learned representations are based on correlations of fixed locality. These issues prevent existing methods from reaching their full potential in terms of prediction performance. Instead, we consider a novel asynchronous message passing approach where information is pushed only along the most relevant edges until convergence. Our proposed algorithm can equivalently be formulated as a single synchronous message passing iteration using a suitable neighborhood function, thus sharing the advantages of existing methods while addressing their central issues. The resulting neural network utilizes a node-adaptive receptive field derived from meaningful sparse node neighborhoods. In addition, by learning and combining node representations over differently sized neighborhoods, our model is able to capture correlations on multiple scales. We further propose variants of our base model with different inductive bias. Empirical results are provided for semi-supervised node classification on five real-world datasets following a rigorous evaluation protocol. We find that our models outperform competitors on all datasets in terms of accuracy with statistical significance. In some cases, our models additionally provide faster runtime.

1 Introduction

As a natural abstraction of real-world entities and their relationships, graphs are widely adopted as a tool for modeling machine learning tasks on relational data. Applications are manifold, including documents classification in citation networks, user recommendations in social networks or function prediction of proteins in biological networks.

Remarkable success has been achieved by recent efforts in formulating deep learning models operating on graph-structured domains. Unsupervised node embedding techniques [33, 39, 7, 15, 34] employ matrix factorization to derive distributed vector space representations for further downstream tasks. In settings where labels are provided, semi-supervised models can be trained end-to-end to improve performance for a given task. In particular, graph neural network models [5, 14, 2] have been established as a de-facto standard for semi-supervised learning on graphs. While spectral methods [6, 10, 28, 5] can be derived from a signal processing point of view, a message passing perspective [11, 26, 18, 21, 16, 14] has proved especially useful due to its flexibility and amenability to highly parallel GPU computation [13]. Further recent works have considered additional edge features [14, 36], attention mechanisms [41, 40, 24], addressed scalability [8, 42] and studied the expressive power of graph neural network models [43, 29].

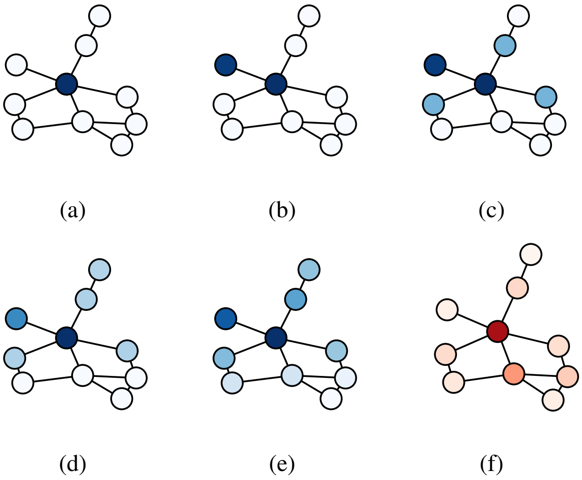


Figure 1: Features of the central node are pushed through the graph until convergence (1a–1e). Equivalently, each node performs a single aggregation step over a node-adaptive neighborhood with importance weights shown for the central node in (1f).

While the above techniques may serve as a basis for modeling further tasks such as link prediction [20, 45] or graph classification [31, 23], we focus on *semi-supervised node classification* in this work. Given a graph $G = (V, E)$, a feature matrix $X \in \mathbb{R}^{n \times d}$ and a label matrix $Y \in \mathbb{R}^{n \times c}$, the goal is to predict labels for a set of unlabeled nodes based on graph topology, node attributes and observed node labels. If no node attributes are available, auxiliary features such as one-hot vectors or node degrees may be used, depending on the task at hand. All graphs considered in the following are undirected, however, extension to directed graphs is straightforward.

Despite their success, existing neural message passing algorithms suffer from several central issues. First, information is pulled indiscriminately from k-hop neighborhoods which will include many irrelevant nodes and miss important ones. In particular, long-range dependencies are modeled ineffectively, since unnecessary messages do not only impede efficiency but additionally introduce noise. Further, interesting correlations might exist on different levels of locality which makes it necessary to consider multi-scale representations. These issues prevent existing neural message passing algorithms from reaching their full potential in terms of prediction per-

[1] Siemens Corporate Technology, Princeton, NJ, USA
jiaxing.pi@siemens.com
[2] Ludwig-Maximilians-Universität München, Munich, Germany
{busch, seidl}@dbs.ifi.lmu.de

Algorithm 1 Synchronous Message Passing

```
Input: Graph $G$, feature matrix $H^{(0)}$
Output: Aggregated feature matrix $H^{(K)}$
  for $k \in [K]$ do
      # Send messages
      for $i \in V$ do
          for $j \in \mathcal{N}_i$ do
              $\phi_{j \to i}^{(k)} = \phi^{(k)} \left( h_i^{(k-1)}, h_j^{(k-1)}, a_{ji} \right)$
          end for
      end for
      # Update node states
      for $i \in V$ do
          $h_i^{(k)} = \gamma^{(k)} \left( h_i^{(k-1)}, \textsc{Aggr}_{j \in \mathcal{N}_i} \, \phi_{j \to i}^{(k)} \right)$
      end for
  end for
```

formance.

To address the above issues, we propose a novel *push*-based neural message passing algorithm which propagates information on demand rather than indiscriminately pulling it from all neighbors. We show that it can be interpreted equivalently as either an asynchronous message passing scheme or a single synchronous message passing iteration over sparse neighborhoods derived from *Approximate Personalized PageRank*. Thereby, each node neighborhood is personalized to its source node, providing a stronger structural bias and resulting in a node-adaptive receptive field. Both views are illustrated in Figure 1. Consequently, our model benefits from the existing synchronous neural message passing framework while providing additional advantages derived from its asynchronous message passing interpretation. In contrast to existing synchronous methods, our model further eliminates the need of stacking multiple message passing layers to reach distant nodes by introducing a suitable neighborhood function. It additionally supports highly efficient training and is able to learn combinations of multi-scale representations.

2 Neural Message Passing Algorithms

Neural message passing algorithms follow a *synchronous* neighborhood aggregation scheme. Starting with an initial feature matrix $H^{(0)} \in \mathbb{R}^{n \times h}$, for K iterations, each node sends a message to each of its neighbors and updates its own state based on the aggregated received messages. Borrowing some notation from [13], we formalize this procedure in Algorithm 1 where $\phi^{(k)}$ is a *message function*, $\textsc{Aggr}$ is a permutation invariant *aggregation function* and $\gamma^{(k)}$ is an *update function*. All of these functions are required to be differentiable.

One of the most simple and widespread representatives of this framework is the *Graph Convolutional Network (GCN)* [21] which can be defined via

$$\phi_{j \to i}^{(k)} = \hat{\tilde{\mathrm{A}}}_{ji} \, W^{(k)} h_j^{(k)} \tag{1}$$

$$\textsc{Aggr}_i^{(k)} = \sum_{j \in \mathcal{N}_i} \phi_{j \to i}^{(k)} \tag{2}$$

$$\gamma_i^{(k)} = q^{(k)} \left(\textsc{Aggr}_i^{(k)} \right) \tag{3}$$

such that

$$H^{(k+1)} = q^{(k)} \left(\hat{\tilde{\mathrm{A}}} \, H^{(k)} W^{(k)} \right) \tag{4}$$

where $H^{(0)} = X$, $q^{(k)}$ is a non-linearity (*ReLU* is used for hidden layers, *softmax* for the final prediction layer), $\hat{\tilde{\mathrm{A}}} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}$ is a symmetrically normalized adjacency matrix with self-loops and $\tilde{A} = A + I$ with degree matrix $\tilde{D}$. Note that due to self-loops each node also aggregates its own features. Normalization preserves the scale of the feature vectors. In each GCN layer, features are transformed and aggregated from direct neighbors as a weighted sum.

While various models which can be formulated in this framework have achieved remarkable performance, a general issue with synchronous message passing schemes is that long-range dependencies in the graph are not modeled effectively. If $\mathcal{N}_i$ denotes the one-hop neighborhood of node i (which is commonly the case), then each message passing iteration expands the receptive field by one hop. For a single node to gather information from another node of distance K, K message passing iterations need to be performed for *all* nodes in the graph. Sending a large number of unnecessary messages does not only result in unnecessary computation but further introduces noise to the learned node features. On the same note, [44] and [25] pointed out an over-smoothing effect. [44] showed that with an increasing number of layers, node importance in a GCN converges to the graph's random walk limit distribution, i.e., all local information is lost.

2.1 Asynchronous Message Passing

Instead of passing messages along all edges in multiple subsequent rounds, one might consider an *asynchronous* propagation scheme where nodes perform state updates and send messages one after another. In particular, pushing natively supports adaptivity, since instead of just pulling information from all neighbors, nodes are able to push and receive important information on demand. This motivates our *push-based message passing framework* (Algorithm 2).

Algorithm 2 Asynchronous Push-based Message Passing

```
Input: Graph $G$, feature matrix $H^{(0)}$
Output: Aggregated feature matrix $H$
  Initialize $\Phi_i$ for all $i \in V$
  while not converged do
      Select next node $i \in V$
      # Update node state
      $h_i \leftarrow \gamma \left( h_i, \Phi_i \right)$
      # Send messages
      for $j \in \mathcal{N}_i$ do
          $\phi_{i \to j} = \phi \left( h_i, h_j, \Phi_i, \Phi_j, a_{ij} \right)$
          $\Phi_j \leftarrow \textsc{Aggr} \left( \Phi_j, \phi_{i \to j} \right)$
      end for
      reset $\Phi_i$
  end while
```

First, it is important to note that each node needs to aggregate incoming messages until it is selected to be updated. For that purpose, we introduce aggregator states $\Phi_i \in \mathbb{R}^h$ which contain novel unprocessed information for each node. After it is used by a node to update its state and it has pushed messages to its neighbors, the aggregator state is reset until the node receives more information and becomes active again. Further note that the aggregator states naturally lend themselves to serve as a basis for selecting the next node and for a convergence criterion, based on the amount of unprocessed information. The functions ϕ, $\textsc{Aggr}$ and γ fulfill the same roles and share the same requirements as their synchronous counterparts. Though not

Algorithm 3 Local Push Message Passing (LPMP)

```
Input: Graph $G$, feature matrix $H^{(0)}$, parameters $\alpha \in (0,1)$, $\varepsilon > 0$
Output: Aggregated feature matrix $H$
  Initialize dense matrix $H = zeros(n, h)$
  for $k \in V$ do
      $\Phi_i = \delta_{i,k} h_i$ for all $i \in V$
      while $\max_{i \in V} ||\Phi_i|| > \varepsilon ||h_k||$ do
          $h_i \leftarrow h_i + \alpha \Phi_i$
          for $j \in \mathcal{N}_i$ do
              $\Phi_j \leftarrow \Phi_j + \frac{1-\alpha}{d_j} \Phi_i$
          end for
          $\Phi_i = 0$
      end while
  end for
```

specifically indicated, in principle, different functions may be used for different iterations.

As a particular instance of this framework, we propose *Local Push Message Passing (LPMP)* (Algorithm 3). For the next update, the node with the largest aggregator state is selected, since it holds the largest amount of unprocessed information. Similarly, convergence is attained if each node has only a small amount of unprocessed information left. Note that all state updates are additive and no learnable transformations are applied in order to effectively treat long-range dependencies and retain flexibility. Feature transformations may be applied before or after propagation. Further, in each iteration of the outer loop, only the features of node k are diffused through the graph in order to avoid excessive smoothing which might occur when multiple features are propagated at the same time over longer distances in the graph. Also, all iterations of the outer loop are independent of each other and can be performed in parallel. Remaining details of the algorithm will be motivated and explained below.

We further wish to point out that the synchronous framework does not consider any notion of convergence but instead introduces a hyper-parameter for the number of message passing iterations. An early work on *Graph Neural Network (GNN)* [35] applies contraction mappings and performs message passing until node states converge to a fixed point. However, neighborhood aggregation is still performed synchronously.

Finally, further instances of the general push-based message passing framework may be considered in future work. We focus on this particular algorithm due to its nice interpretation in terms of existing push algorithms (as detailed below), its favorable properties and since we observed it to perform well in practice.

3 Pushing Networks

The LPMP algorithm described above is inspired by local *push* algorithms for computation of *Approximate Personalized PageRank (APPR)* [17, 3] and, in particular, we will show in the following how it can be equivalently described as a single synchronous message passing iteration using sparse APPR neighborhoods. Thus, the proposed message passing scheme effectively combines the advantages of existing synchronous algorithms with the benefits of asynchronous message passing described above.

3.1 Personalized Node Neighborhoods

Personalized PageRank (PPR) refers to a localized variant of *PageRank* [32] where random walks are restarted only from a certain set of nodes. We consider the special case in which the starting distribution is a unit vector, i.e., when computing PPR-vector π_i of node i, walks are always restarted at i itself. Formally, π_i can be defined as the solution of the linear system

$$\pi_i = \alpha e_i + (1-\alpha)\pi_i A_{rw} \tag{5}$$

where $e_i \in \mathbb{R}^n$ denotes the ith unit vector, $A_{rw} = D^{-1}A$ denotes the random walk transition matrix of G, and the restart probability $\alpha \in (0,1)$ controls the locality, where a larger value leads to stronger localization. The PPR vectors for all nodes can be stored as rows of a PPR-matrix $\Pi \in \mathbb{R}^{n \times n}$. Intuitively, π_{ij} corresponds to the probability that a random walk starting at i stops at j where the expected length of the walk is controlled by α. The vector π_i can be interpreted as an importance measure for node i over all other nodes where π_{ij} measures the importance of j for i. Since these measures are not sparse and global computation of Π would require $\mathcal{O}(n^2)$ operations, we consider local computation of APPR instead. In particular, we refer to the *Reverse Local Push* algorithm [1], since it comes with several useful theoretical properties. Complexity of computing the whole matrix P is reduced to $\mathcal{O}({}^{n}\!/_{\alpha\varepsilon})$ [1], i.e., linear in the number of nodes. The parameter $\varepsilon > 0$ controls the quality of approximation, sparsification and runtime where a larger value leads to sparser solutions. For a more in-depth discussion, we refer to [1].

3.2 PushNet

Based on the above neighborhood function, we propose the following neural message passing algorithm:

Definition 1 (PushNet) *Let f and g be MLPs parametrized by θ_f and θ_g, respectively, h_1, h_2, h_3 denote hidden dimensions, $P = \left[P^{(\alpha_1)}, \ldots, P^{(\alpha_K)}\right] \in \mathbb{R}^{K \times n \times n}$ be a tensor storing precomputed APPR matrices for different scales $\alpha_1 \geq \cdots \geq \alpha_K$ and* AGGR *denote a differentiable* scale aggregation function. *Given input features $X \in \mathbb{R}^{n \times d}$, the layers of* PushNet *are defined as*

$$H^{(0)} = f\left(X; \theta_f\right) \qquad \in \mathbb{R}^{n \times h_1} \tag{6}$$

$$H^{(1)} = PH^{(0)} \qquad \in \mathbb{R}^{K \times n \times h_1} \tag{7}$$

$$H^{(2)} = \text{AGGR}\left(H^{(1)}\right) \qquad \in \mathbb{R}^{n \times h_2} \tag{8}$$

$$H^{(3)} = g\left(H^{(2)}; \theta_g\right) \qquad \in \mathbb{R}^{n \times h_3} \tag{9}$$

In most cases, $h_3 = c$, such that $H^{(3)}$ provides the final predictions for each node over c classes. In general, PushNet might also be applied to different graph learning problems such as graph classification, where learned node representations are pooled and labels are predicted for whole graphs. However, we leave these further applications to future work.

To draw the connection between synchronous and asynchronous message passing, we show that the base variant of PushNet with no feature transformations and a single scale α is equivalent to LPMP (Algorithm 3):

Theorem 1 *Let $\alpha \in (0,1)$ and $\varepsilon > 0$ be fixed, f, g,* AGGR *be identity functions and $K = 1$. Then $H^{(3)} = H$ where $H^{(3)} = PX$ is the output of PushNet and H is the output of LPMP.*

The main idea is that instead of propagating features directly as in LPMP, we can first propagate scalar importance weights as in Reverse Local Push and then propagate features in a seconds step. Thus, all discussion on LPMP are directly applicable to PushNet, including adaptivity, effective treatment of long-range dependencies and avoidance of over-smoothing. We wish to point out that an additional interpretation of adaptivity can be derived from the perspective of PushNet: APPR-induced neighborhoods of different nodes are sparse and directly exclude irrelevant nodes from consideration, in contrast to commonly used k-hop neighborhoods. In this sense, APPR is adaptive to the particular source node. To the best of our knowledge, no existing neural message passing algorithm shares this property.

In practice, it is favorable to not propagate features using LPMP, but to pre-compute APPR matrices such that features are propagated only once along all non-zero APPR entries and there is no need to propagate gradients back over long paths of messages. Thus, PushNet benefits from the existing synchronous neural message passing framework while providing additional advantages derived from its asynchronous interpretation.

3.3 Learning Multi-Scale Representations

Additional properties of PushNet compared to LPMP include feature transformations f and g which may be applied before and after feature propagation. Since the optimal neighborhood size cannot be assumed to be the same for each node and patterns might be observed at multiple scales, we additionally propagate features over different localities by varying the restart probability α. The multi-scale representations are then aggregated per node into a single vector such that the model learns to combine different scales for a given node. In particular, we consider the following scale aggregation functions:

- **sum**: Summation of multi-scale representations. Intuitively, sum-aggregation corresponds to an unnormalized average with uniform weights attached to all scales.

$$\textsc{Aggr}\left(H^{(1)}\right) = \sum_{k\in[K]} P^{(\alpha_k)} H^{(0)} \in \mathbb{R}^{n\times h_1} \tag{10}$$

  Note that due to distributivity, PushNet with sum aggregation reduces to propagation with a single matrix $P = \sum_{k\in[K]} P^{(\alpha_k)}$, i.e., features can be propagated and additively combined over an arbitrary number of different scales at the cost of only a single propagation. Thereby, the non-zero entries in P are given by $\text{nz}\,(P) = \bigcup_{k\in[K]} \text{nz}\left(P^{(\alpha_k)}\right)$. However, usually the number of non-zero entries $\text{nnz}\,(P)$ will be close to $\text{nnz}\left(P^{(\alpha_K)}\right)$, since nodes considered at a smaller scale will most often also be considered at a larger scale. Thus, complexity will be dominated by the largest scale considered.
- **max**: Element-wise maximum of multi-scale representations. The most informative scale is selected for each feature individually. This way, different features may correspond to more local or more global properties.

$$\textsc{Aggr}\left(H^{(1)}\right) = \max_{k\in[K]} P^{(\alpha_k)} H^{(0)} \in \mathbb{R}^{n\times h_1} \tag{11}$$

- **cat**: Concatenation of multi-scale representations. Scale combination is learned in subsequent layers. The implied objective is to learn a scale aggregation function which is globally optimal for all nodes.

$$\textsc{Aggr}\left(H^{(1)}\right) = \big\Vert_{k\in[K]} P^{(\alpha_k)} H^{(0)} \in \mathbb{R}^{n\times (K\cdot h_1)} \tag{12}$$

3.4 The PushNet Model Family

We wish to point out several interesting special cases of our model. In our default setting, prediction layers will always be dense with a *softmax* activation. If hidden layers are used, we use a single dense layer with *ReLU* activation.

- **PushNet.** The general case in which f and g are generic MLPs. As per default, f is a single dense hidden layer and g is a dense prediction layer.
- $\mathbf{f = id}$. No feature transformation is performed prior to propagation. In this case, $H^{(2)}$ needs to be computed only once and can then be cached, making learning extremely efficient. The following sub-cases are of particular interest:
  - **PushNet-PTP.** The sequence of operations is ”push – transform – predict”. In this case, g is a generic MLP, consisting of 2 layers per default.
  - **PushNet-PP.** The model performs operations ”push – predict” and uses no hidden layers. Predictions can be interpreted as the result of logistic regression on aggregated features. This version is similar to SGC [42] with the difference that SGC does not consider multiple scales and propagates over k-hop neighborhoods.
  - **LPMP.** In this setting, $g = id$ and $K = 1$, i.e., no feature transformations are performed and features are aggregated over a single scale. This setting corresponds to LPMP, cf. Theorem 1. Note that this model describes only feature propagation, no actual predictions are made.
- **PushNet-TPP.** The setting is $h_1 = c$ and $g = id$, such that the model first predicts class labels for each node and then propagates the predicted class labels. This setting is similar to APPNP [22] but with some important differences. APPNP considers only a single fixed scale and does not propagate over APPR neighborhoods. Instead, multiple message passing layers are stacked to perform a power iteration approximation of PPR. The resulting receptive field is restricted to k-hop neighborhoods. Note that *cat* aggregation is not applicable here.

3.5 Comparison with Existing Neural Message Passing Algorithms

Existing message passing algorithms have explored different concepts of node importance, i.e., weights used in neighborhood aggregation. While GCN [21] and other GCN-like models use normalized adjacency matrix entries in each layer, *Simplified Graph Convolution (SGC)* [42] aggregates nodes over k-hop neighborhoods in a single iteration using a k-step random walk matrix. *Approximate Personalized Propagation of Neural Predictions (APPNP)* [22] also relies on a k-step random walk matrix but with restarts which can be equivalently interpreted as a power iteration approximation of the PPR matrix. *Graph Attention Network (GAT)* [41] learns a similarity function which computes a pairwise importance score given two nodes' feature vectors. All of the above methods aggregate features over fixed k-hop neighborhoods. PushNet on the other hand aggregates over sparse APPR neighborhoods using the corresponding importance scores.

Multi-scale representations have been considered in *Jumping Knowledge Networks (JK)* [44] where intermediate representations of a GCN or GAT base network are combined before prediction. The original intention was to avoid over-smoothing by introducing these

	$\|\mathbf{V}\|$	$\|\mathbf{E}\|$	$\mathbf{d}$	$\mathbf{c}$	$\mathbf{avgSP}$	$\mathbf{maxSP}$
CiteSeer	2120	3679	3703	6	9.33	28
Cora	2485	5069	1433	7	6.31	19
PubMed	19717	44324	500	3	6.34	18
Coauthor CS	18333	81894	6805	15	5.43	24
Coauthor Physics	34493	247962	8415	5	5.16	17

Table 1: Dataset statistics including average and maximum shortest path lengths considering only the largest connected component of each graph.

skip connections. Similarly, [27] concatenate propagated features at multiple selected scales. LD [12] uses APPR to compute local class label distributions at multiple scales and proposes different combinations but is limited to unattributed graphs. PushNet varies the restart probability in APPR to compute multi-scale representations and combines them using simple and very efficient aggregation functions. APPR-Roles [4] also employs APPR but to compute structural node embeddings in an unsupervised setting. The idea of performing no learnable feature transformations prior to propagation was explored already in SGC [42]. It allows for caching propagated features, resulting in very efficient training, and is also used by PushNet-PTP and PushNet-PP. LD [12] explored the idea of propagating class labels instead of latent representations. Similarly, APPNP [22] and PushNet-TPP propagate predicted class labels.

To the best of our knowledge, the only existing work considering asynchronous neural message passing is SSE [9]. Compared to PushNet, SSE is pull-based, i.e., in each iteration a node pulls features from all neighbors and updates its state, until convergence to steady node states. To make learning feasible, stochastic training is necessary. Further, the work focuses on learning graph algorithms for different tasks and results for semi-supervised node classification were not very competitive. In contrast, PushNet offers very fast deterministic training and adaptive state updates due to a push-based approach.

4 Experiments

We compare PushNet and its variants against six state-of-the-art models, GCN [21], GAT [41], JK [44] with base model GCN and GAT, SGC [42], *Graph Isomorphism Network (GIN)* [43] and APPNP [22] on five established node classification benchmark datasets. For better comparability, all models were implemented using *PyTorch Geometric* [13] [3] and trained on a single NVIDIA GeForce GTX 1080 Ti GPU.

4.1 Datasets

Experiments are performed on semi-supervised text classification benchmarks. In particular, we consider three citation networks, *CiteSeer* and *Cora* from [37] and *PubMed* from [30], and two coauthorship networks, *Coauthor CS* and *Coauthor Physics* from [38]. Statistics of these datasets are summarized in Table 1.

4.2 Experimental Setup

For the sake of an unbiased and fair comparison, we follow a rigorous evaluation protocol, similarly as in [38] and [22]. [4] We restrict all graphs to their largest connected components and $L1$-normalize all feature vectors. Self-loops are added and different normalizations are applied to the adjacency matrices individually for each method as proposed by the respective authors. For each dataset, we sample 20 nodes per class for training and 500 nodes for validation. The remaining nodes are used as test data. Each model is evaluated on 20 random data splits with 5 random initializations, resulting in 100 runs per model and dataset. Using the same random seed for all models ensures that all models are evaluated on the same splits.

	Hidden size	**Learning rate**	**Dropout**	L_2 **reg. strength**
GCN	64	0.01	0.5	0.001
GAT	64	0.01	0.6	0.001
SGC	—	0.01	—	0.0001
GIN	64	0.01	0.6	—
JK-GCN (cat)	64	0.01	0.4	0.001
JK-GAT (cat)	64	0.01	0.4	0.001
APPNP ($\alpha = 0.1, K = 10$)	64	0.01	0.5	0.01
PushNet (sum)	64	0.005	0.5	0.01
PushNet-PTP (sum)	64	0.005	0.3	0.1
PushNet-PP (sum)	—	0.01	0.6	0.001
PushNet-TPP (sum)	32	0.01	0.5	0.01

Table 2: Optimal hyper-parameters for all models as determined by a grid search on CiteSeer and Cora. Values in parentheses indicate optimal model-specific hyper-parameters.

Model architectures including sequences and types of layers, activation functions, locations of dropout and $L2$-regularization are fixed as recommended by the respective authors. All remaining hyperparameters are optimized per model by selecting the parameter combination with best average accuracy on CiteSeer and Cora validation sets. Final results are reported only for the test sets using optimal parameters. All models are trained with *Adam* [19] using default parameters and early stopping based on validation accuracy and loss as in [41] with a patience of 100 for a maximum of 10000 epochs.

For all PushNet variants, we fix the architecture as described in the previous section. Dropout is applied to all APPR matrices and to the inputs of all dense layers. However, for PushNet-PTP and PushNet-PP dropout is only applied after propagation, such that propagated features can be cached. $L2$-regularization is applied to all dense layers. As a default setting, we consider three different scales $\alpha \in \{0.2, 0.1, 0.05\}$ and $\varepsilon = 1e-5$. Due to memory constraints, we use $\varepsilon = 1e-4$ on Physics dataset for all PushNet variants and on CS and PubMed datasets for PushNet and PushNet-TPP. We further add self-loops to the adjacency matrices and perform symmetric normalization as in GCN. All APPR-matrices are $L1$-normalized per row. We use the following parameter grid for tuning hyper-parameters of all models:

- Number of hidden dimensions: [8, 16, 32, 64]
- Learning rate: [0.001, 0.005, 0.01]
- Dropout probability: [0.3, 0.4, 0.5, 0.6]
- Strength of L2-regularization: [1e-4, 1e-3, 1e-2, 1e-1]

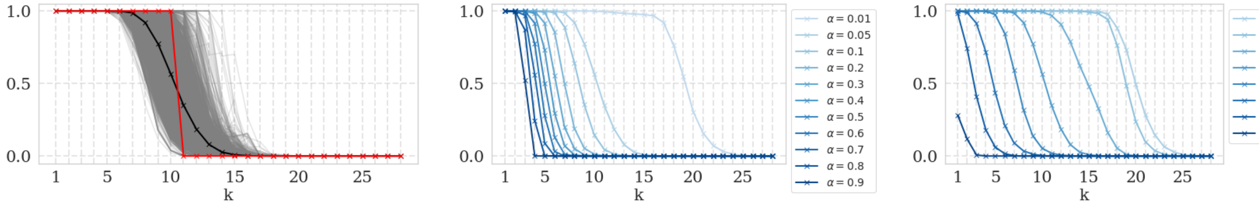
Except for APPNP, all competitors use $K = 2$ propagation layers. JK and GIN use an additional dense layer for prediction. For GAT layers, the number of attention heads is fixed to 8. Optimal hyperparameters for all models are reported in Table 2.

4.3 Node Classification Accuracy

Accuracy/micro-F1 scores for all datasets are provided in Table 3. It can be observed that our models consistently provide best results on

[3] https://github.com/rusty1s/pytorch_geometric

[4] Note that results for competing methods might differ from those reported in related work due to a different experimental setup.



(a) CiteSeer ($\alpha = 0.05$, $\varepsilon = 1e-5$)

(b) CiteSeer ($\varepsilon = 1e-5$)

(c) CiteSeer ($\alpha = 0.05$)

Figure 2: Fraction of k-hop neighbors contained in APPR-neighborhoods for CiteSeer. (2a) shows all APPR neighborhoods for a fixed setting, including mean and the closest k-neighborhood for reference. (2b) and (2c) demonstrate localization depending on α and sparsification depending on ε, respectively.

	GCN	**GAT**	**JK-GCN**	**JK-GAT**	**SGC**	**GIN**	**APPNP**	**PushNet**	**PushNet-PTP**	**PushNet-PP**	**PushNet-TPP**
CiteSeer	72.82 ± 1.48	73.82 ± 1.35	71.09 ± 1.66	71.76 ± 1.27	73.91 ± 1.30	70.81 ± 1.61	74.36 ± 1.44	75.08 ± 0.99	**75.19 ± 1.15**	75.17 ± 1.32	75.01 ± 1.11
Cora	81.07 ± 1.43	82.12 ± 1.41	79.57 ± 1.63	80.10 ± 1.52	80.13 ± 2.15	80.24 ± 1.54	83.58 ± 1.03	84.12 ± 1.08	83.41 ± 1.24	81.52 ± 1.40	**84.23 ± 1.26**
PubMed	78.29 ± 1.48	78.21 ± 1.60	77.23 ± 2.01	77.59 ± 2.25	77.00 ± 1.78	77.19 ± 1.75	79.61 ± 2.98	79.80 ± 1.39	**80.22 ± 1.27**	77.52 ± 2.05	80.10 ± 1.33
Coauthor CS	91.64 ± 0.62	90.20 ± 0.75	91.60 ± 0.54	92.20 ± 0.43	91.27 ± 0.58	91.46 ± 0.54	91.10 ± 1.12	92.40 ± 0.52	92.37 ± 0.40	91.04 ± 0.76	**92.54 ± 0.34**
Coauthor Physics	93.42 ± 0.63	93.43 ± 0.50	93.49 ± 0.56	*o.o.m.*	*o.o.m.*	93.79 ± 0.49	93.96 ± 0.45	94.01 ± 0.53	93.97 ± 0.48	93.67 ± 0.55	**94.09 ± 0.47**

Table 3: Accuracy/micro-F1 scores on semi-supervised node classification datasets in terms of mean and standard deviation over 100 independent runs. JK-GAT and SGC are out of GPU memory on the largest dataset, Coauthor Physics.

	CiteSeer	**Cora**	**PubMed**	**Coauthor CS**	**Coauthor Physics**
Accuracy	4.18e-08	3.37e-07	1.20e-02	4.62e-11	3.45e-03
Macro-F1	3.91e-10	2.79e-04	2.22e-02	9.22e-12	2.12e-07

Table 4: P-values according to a Wilcoxon signed-rank test comparing the best of our models with the best competitor on all datasets with respect to accuracy/micro-F1 and macro-F1.

all datasets and that the strongest model, PushNet-TPP, outperforms all competitors on all datasets. Improvements of our best model compared to the best competing model are statistically significant with $P < .05$ on all datasets according to a Wilcoxon signed-rank test. [5] P-values for all datasets are reported in Table 4. On CiteSeer and PubMed, PushNet-PTP is able to push performance even further. PushNet with feature transformations before and after propagation is less performant but still outperforms all competitors on all datasets. PushNet-PP, the most simple of our models, performs worst as expected. However, it is still competitive, outperforming all competitors on CiteSeer. Boxplots shown in Figure 3 indicate that our models generally exhibit small variance and are less prone to produce outlier scores.

We argue that improvements over existing methods are primarily due to push-based propagation. Figure 2a compares APPR neighborhoods with k-hop neighborhoods in terms of the fraction of k-neighbors considered. It can be seen that k-neighborhoods used by competitors draw a sharp artificial boundary while APPR adaptively selects nodes from larger neighborhoods and discards nodes from smaller ones, *individually for each source node*. Visually, deviations left to the boundary correspond to discarded irrelevant nodes, while deviations on the right hand side indicate additional nodes beyond the receptive field of competitors that can be leveraged by our method. Stacking more message passing layers to reach these nodes would degrade performance due to overfitting as demonstrated in [44] and [25].

Among the competing methods, APPNP performs best in general, providing best baseline performance on all datasets but CS where JK-GAT achieves best results. GAT outperforms GCN on three datasets, CiteSeer, Cora and Physics. JK performs worse than its respective basemodel in most cases. Similar observations were already made in [22]. SGC mostly performs worse than GCN due to its simplicity, outperforming it only on CiteSeer. GIN also provides worse results than GCN in most cases, possibly due to overfitting caused by larger model complexity. It outperforms GCN only on Physics, providing results similar to APPNP.

On CiteSeer, models using cached features perform very well, even the simple models SGC and PushNet-PP which effectively perform linear regression on propagated raw features provide superior performance. On the remaining datasets, the additional feature transformation provided by PushNet-PTP is necessary to guarantee high accuracy.

Macro-F1 scores reveal similar insights and are ommitted due to space constraints.

4.4 Runtime

Figure 4 compares all methods based on average runtime per epoch and accuracy. [6] SGC has lowest runtime on all datasets but runs out of memory on Physics since it propagates raw features over k-hop neighborhoods. PushNet-PP performs second fastest, followed by PushNet-PTP which generally provides a good tradeoff between runtime and accuracy. PushNet and PushNet-TPP are slower than competitors but still provide comparable runtime at a superior level of accuracy. Among the competitors, APPNP, GAT and JK-GAT require most computation time. JK-GAT also runs out of memory on Physics.

[5] In fact, results are significant with $P < 0.01$ on all datasets except PubMed.

[6] We note that for APPNP and all PushNet variants, (A)PPR matrix computation is not included in runtime analysis such that runtime comparison is solely based on propagation, transformation and prediction for all compared models. We consider (A)PPR computation as a preprocessing step, since it is only required to be performed once per graph and can then be reused by all models for this graph. Computation is very fast for each of the considered graphs and can be performed on CPU.

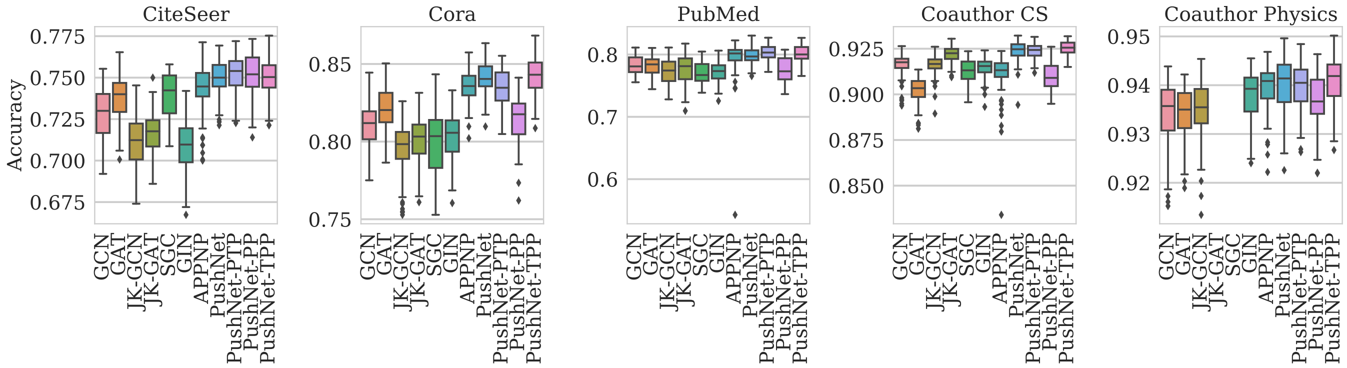


Figure 3: Accuracy/micro-F1 scores on semi-supervised node classification datasets aggregated over 100 independent runs.

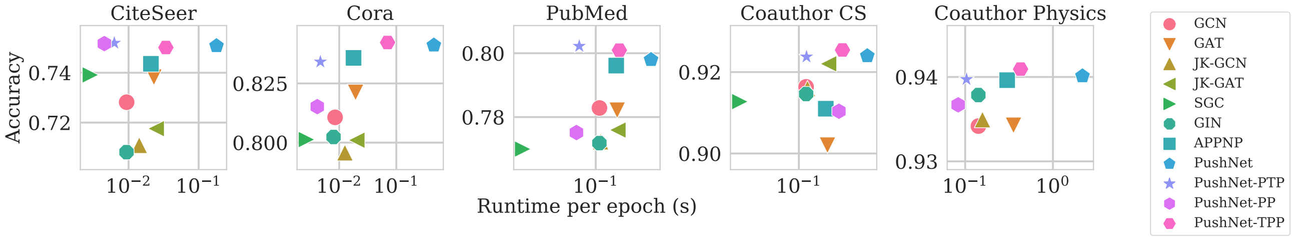


Figure 4: Comparison w.r.t. average runtime per epoch in seconds vs. average accuracy.

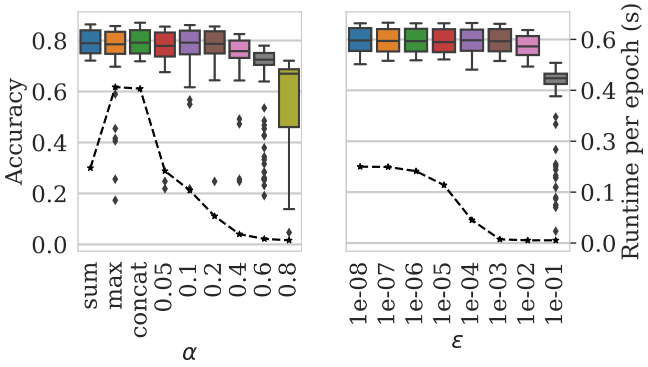


Figure 5: Effect of varying α for fixed $\varepsilon = 1e-5$ (left) and varying ε for fixed $\alpha \in \{0.05, 0.1, 0.2\}$ with *sum* aggregation (right). Dashed lines indicate average runtime per epoch in seconds.

4.5 Influence of Locality

To study the influence of the locality parameter α on the performance of our models, we run experiments with our base model PushNet with various single α values and different aggregations of multiple values. Figure 2b illustrates how the fraction of k-neighbors considered for propagation varies with α. Generally, a larger value leads to stronger localization and the shape gets closer to a step function as for k-neighborhoods. Figure 5 additionally shows the average performance on CiteSeer and Cora. For single α, small values in $\{0.05, 0.1, 0.2\}$ achieve best accuracy. For larger α, runtime drops considerably but at the cost of decreased accuracy and larger variance. Among multi-scale aggregations, *sum* performs best. It slightly improves the performance over single alphas and provides additional robustness, producing smaller variance and no outlier scores. Runtime is very close to the smallest single α considered. The remaining aggregation functions lead to increased runtime and *max* does not even lead to an increase of accuracy.

4.6 Influence of Sparsity

Similarly as α, the approximation threshold ε controls the effective neighborhood size considered for propagation. We study the effect on our models with a similar setup as above. Figure 2c illustrates how a larger value of ε leads to stronger sparsification of APPR neighborhoods. Variation leads to a shift of the curve, indicating that neighbors with small visiting probabilities are discarded mostly from k-neighborhoods with moderate or large k. Figure 5 shows that accuracy remains relatively stable on CiteSeer and Cora, decreasing monotonically for increasing ε. Simultaneously, runtime decreases steadily. While smaller ε provide marginally better accuracy, our results suggest that ε may be increased safely to allow for faster runtime and to account for limited GPU memory.

5 Conclusion

We presented a novel push-based asynchronous neural message passing algorithm which allows for efficient feature aggregation over adaptive node neighborhoods. A multi-scale approach additionally leverages correlations on increasing levels of locality and variants of our model capture different inductive bias. Semi-supervised node classification experiments on five real-world benchmark datasets exhibit consistent improvements of our models over all competitors with statistical significance, demonstrating the effectiveness of our approach. Ablation studies investigate the influence of varying locality and sparsity parameters as well as combinations of multi-scale representations. In future work, we intend to investigate additional instances of the push-based message passing framework, extensions to dynamic graphs and applications to further tasks such as link prediction and graph classification.

ACKNOWLEDGEMENTS

This work was done during an internship at CT RDA BAM IBI-US, Siemens Corporate Technology, Princeton, NJ, USA.

REFERENCES

[1] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng, 'Local computation of pagerank contributions', *International Workshop on Algorithms and Models for the Web-Graph*, (2007).

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al., 'Relational inductive biases, deep learning, and graph networks', *arXiv preprint arXiv:1806.01261*, (2018).

[3] Pavel Berkhin, 'Bookmark-coloring algorithm for personalized pagerank computing', *Internet Mathematics*, (2006).

[4] Felix Borutta, Julian Busch, Evgeniy Faerman, Adina Klink, and Matthias Schubert, 'Structural graph representations based on multiscale local network topologies', *WI*, (2019).

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst, 'Geometric deep learning: going beyond euclidean data', *IEEE Signal Processing Magazine*, (2017).

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, 'Spectral networks and locally connected networks on graphs', *ICLR*, (2014).

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu, 'Grarep: Learning graph representations with global structural information', *CIKM*, (2015).

[8] Jie Chen, Tengfei Ma, and Cao Xiao, 'Fastgcn: fast learning with graph convolutional networks via importance sampling', *ICLR*, (2018).

[9] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song, 'Learning steady-states of iterative algorithms over graphs', *ICML*, (2018).

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, 'Convolutional neural networks on graphs with fast localized spectral filtering', *NeurIPS*, (2016).

[11] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams, 'Convolutional networks on graphs for learning molecular fingerprints', *NeurIPS*, (2015).

[12] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert, 'Semi-supervised learning on graphs based on local label distributions', *arXiv preprint arXiv:1802.05563*, (2018).

[13] Matthias Fey and Jan Eric Lenssen, 'Fast graph representation learning with pytorch geometric', *arXiv preprint arXiv:1903.02428*, (2019).

[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl, 'Neural message passing for quantum chemistry', *ICML*, (2017).

[15] Aditya Grover and Jure Leskovec, 'node2vec: Scalable feature learning for networks', *SIGKDD*, (2016).

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec, 'Inductive representation learning on large graphs', *NeurIPS*, (2017).

[17] Glen Jeh and Jennifer Widom, 'Scaling personalized web search', *WWW*, (2003).

[18] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley, 'Molecular graph convolutions: moving beyond fingerprints', *Journal of computer-aided molecular design*, (2016).

[19] Diederik P Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', *ICLR*, (2015).

[20] Thomas N Kipf and Max Welling, 'Variational graph auto-encoders', *NeurIPS Bayesian Deep Learning Workshop*, (2016).

[21] Thomas N Kipf and Max Welling, 'Semi-supervised classification with graph convolutional networks', *ICLR*, (2017).

[22] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann, 'Predict then propagate: Graph neural networks meet personalized pagerank', *ICLR*, (2019).

[23] John Boaz Lee, Ryan Rossi, and Xiangnan Kong, 'Graph classification using structural attention', *SIGKDD*, (2018).

[24] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh, 'Attention models in graphs: A survey', *TKDD*, (2019).

[25] Qimai Li, Zhichao Han, and Xiao-Ming Wu, 'Deeper insights into graph convolutional networks for semi-supervised learning', *AAAI*, (2018).

[26] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel, 'Gated graph sequence neural networks', *ICLR*, (2016).

[27] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel, 'Lanczosnet: Multi-scale deep graph convolutional networks', *ICLR*, (2019).

[28] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein, 'Geometric deep learning on graphs and manifolds using mixture model cnns', *CVPR*, (2017).

[29] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe, 'Weisfeiler and leman go neural: Higher-order graph neural networks', *AAAI*, (2019).

[30] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU, 'Query-driven active surveying for collective classification', *MLG*, (2012).

[31] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov, 'Learning convolutional neural networks for graphs', *ICML*, (2016).

[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd, 'The pagerank citation ranking: Bringing order to the web.', Technical report, Stanford InfoLab, (1999).

[33] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, 'Deepwalk: Online learning of social representations', *SIGKDD*, (2014).

[34] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang, 'Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec', *WSDM*, (2018).

[35] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, 'The graph neural network model', *IEEE Transactions on Neural Networks*, (2009).

[36] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling, 'Modeling relational data with graph convolutional networks', *ESWC*, (2018).

[37] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad, 'Collective classification in network data', *AI magazine*, (2008).

[38] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann, 'Pitfalls of graph neural network evaluation', *NeurIPS Relational Representation Learning Workshop*, (2018).

[39] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei, 'Line: Large-scale information network embedding', *WWW*, (2015).

[40] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li, 'Attention-based graph neural network for semi-supervised learning', *arXiv preprint arXiv:1803.03735*, (2018).

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, 'Graph attention networks', *ICLR*, (2018).

[42] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger, 'Simplifying graph convolutional networks', *ICML*, (2019).

[43] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, 'How powerful are graph neural networks?', *ICLR*, (2019).

[44] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Kenichi Kawarabayashi, and Stefanie Jegelka, 'Representation learning on graphs with jumping knowledge networks', *ICML*, (2018).

[45] Muhan Zhang and Yixin Chen, 'Link prediction based on graph neural networks', *NeurIPS*, (2018).