Position-aware Graph Neural Networks

Jiaxuan You [1] Rex Ying [1] Jure Leskovec [1]

Abstract

Learning node embeddings that capture a node's position within the broader graph structure is crucial for many prediction tasks on graphs. However, existing Graph Neural Network (GNN) architectures have limited power in capturing the position/location of a given node with respect to all other nodes of the graph. Here we propose *Position-aware Graph Neural Networks (P-GNNs)*, a new class of GNNs for computing position-aware node embeddings. P-GNN first samples sets of anchor nodes, computes the distance of a given target node to each anchor-set, and then learns a non-linear distance-weighted aggregation scheme over the anchor-sets. This way P-GNNs can capture positions/locations of nodes with respect to the anchor nodes. P-GNNs have several advantages: they are inductive, scalable, and can incorporate node feature information. We apply P-GNNs to multiple prediction tasks including link prediction and community detection. We show that P-GNNs consistently outperform state of the art GNNs, with up to 66% improvement in terms of the ROC AUC score.

1. Introduction

Learning low-dimensional vector representations of nodes in graphs (Hamilton et al., 2017b) has led to advances on tasks such as node classification (Kipf & Welling, 2017), link prediction (Grover & Leskovec, 2016), graph classification (Ying et al., 2018b) and graph generation (You et al., 2018b), with successful applications across domains such as social and information networks (Ying et al., 2018a), chemistry (You et al., 2018a), and biology (Zitnik & Leskovec, 2017).

Node embedding methods can be categorized into Graph Neural Networks (GNNs) approaches (Scarselli et al., 2009), matrix-factorization approaches (Belkin & Niyogi, 2002), and random-walk approaches (Perozzi et al., 2014). Among these, GNNs are currently the most popular paradigm, largely owing to their efficiency and inductive learning capability (Hamilton et al., 2017a). By contrast, random-walk approaches (Perozzi et al., 2014; Grover & Leskovec, 2016) are limited to transductive settings and cannot incorporate node attributes. In the GNN framework, the embedding of a node is computed by a GNN layer aggregating information from the node's network neighbors via non-linear transformation and aggregation functions (Battaglia et al., 2018). Long-range node dependencies can be captured via stacking multiple GNN layers, allowing the information to propagate for multiple-hops (Xu et al., 2018).

However, the key limitation of existing GNN architectures is that they fail to capture the *position/location* of the node within the broader context of the graph structure. For example, if two nodes reside in very different parts of the graph but have topologically the same (local) neighbourhood structure, they will have identical GNN structure. Therefore, the GNN will embed them to the same point in the embedding space (we ignore node attributes for now). Figure 1 gives an example where a GNN cannot distinguish between nodes v_1 and v_2 and will always embed them to the same point because they have isomorphic network neighborhoods. Thus, GNNs will never be able to classify nodes v_1 and v_2 into different classes because from the GNN point of view they are indistinguishable (again, not considering node attributes). Researchers have spotted this weakness (Xu et al., 2019) and developed heuristics to fix the issue: augmenting node features with one-hot encodings (Kipf & Welling, 2017), or making GNNs deeper (Selsam et al., 2019). However, models trained with one-hot encodings cannot generalize to unseen graphs, and arbitrarily deep GNNs still cannot distinguish structurally isomorphic nodes (Figure 1).

Here we propose *Position-aware Graph Neural Networks (P-GNNs)*, a new class of Graph Neural Networks for computing node embeddings that incorporate a node's positional information with respect to all other nodes in the network, while also retaining inductive capability and utilizing node features. Our key observation is that node position can be captured by a low-distortion embedding by quantifying the distance between a given node and a set of anchor nodes. Specifically, P-GNN uses a sampling strategy with theoreti-

[1] Department of Computer Science, Stanford University, Stanford, CA, USA. Correspondence to: Jiaxuan You <jiaxuan@cs.stanford.edu>, Jure Leskovec <jure@cs.stanford.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

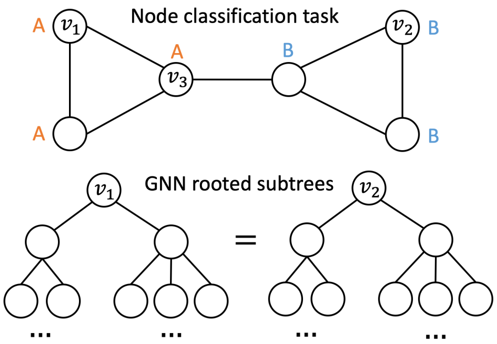

Figure 1. Example graph where GNN is not able to distinguish and thus classify nodes v_1 and v_2 into different classes based on the network structure alone. (Note we do not consider node features.) Each node is labeled based on its label A or B, and effective node embedding should be able to learn to distinguish nodes v_1 and v_2 (that is, embed them into different points in the space). However, GNNs, regardless of depth, will *always* assign the same embedding to both nodes, because the two nodes are symmetric/isomorphic in the graph, and their GNN rooted subtrees used for message aggregation are the same. In contrast, P-GNNs can break the symmetry by using v_3 as the anchor-set, then the shortest path distances (v_1, v_3) and (v_2, v_3) are different and nodes v_1 and v_2 can thus be distinguished.

cal guarantees to choose k random subsets of nodes called *anchor-sets*. To compute a node's embedding, P-GNN first samples multiple anchor-sets in each forward pass, then learns a non-linear aggregation scheme that combines node feature information from each anchor-set and weighs it by the distance between the node and the anchor-set. Such aggregations can be naturally chained and combined into multiple layers to enhance model expressiveness. Bourgain theorem (Bourgain, 1985) guarantees that only $k = O(\log^2 n)$ anchor-sets are needed to preserve the distances in the original graph with low distortion.

We demonstrate the P-GNN framework in various real-world graph-based prediction tasks. In settings where node attributes are not available, P-GNN's computation of the k dimensional distance vector is inductive across different node orderings and different graphs. When node attributes are available, a node's embedding is further enriched by aggregating information from all anchor-sets, weighted by the k dimensional distance vector. Furthermore, we show theoretically that P-GNNs are more general and expressive than traditional message-passing GNNs. In fact, message-passing GNNs can be viewed as special cases of P-GNNs with degenerated distance metrics and anchor-set sampling strategies. In large-scale applications, computing distances between nodes can be prohibitively expensive. Therefore, we also propose P-GNN-Fast which adopts approximate node distance computation. We show that P-GNN-Fast has the same computational complexity as traditional GNN models while still preserving the benefits of P-GNN.

We apply P-GNNs to 8 different datasets and several different prediction tasks including link prediction and community detection[1]. In all datasets and prediction tasks, we show that P-GNNs consistently outperforms state of the art GNN variants, with up to 66% AUC score improvement.

2. Related Work

Existing GNN models belong to a family of graph message-passing architectures that use different aggregation schemes for a node to aggregate feature messages from its neighbors in the graph: Graph Convolutional Networks use mean pooling (Kipf & Welling, 2017); GraphSAGE concatenates the node's feature in addition to mean/max/LSTM pooled neighborhood information (Hamilton et al., 2017a); Graph Attention Networks aggregate neighborhood information according to trainable attention weights (Velickovic et al., 2018); Message Passing Neural Networks further incorporate edge information when doing the aggregation (Gilmer et al., 2017); And, Graph Networks further consider global graph information during aggregation (Battaglia et al., 2018). However, all these models focus on learning node embeddings that capture local network structure around a given node. Such models are at most as powerful as the WL graph isomorphism test (Xu et al., 2019), which means that they cannot distinguish nodes at symmetric/isomorphic positions in the network (Figure 1). That is, without relying on the node feature information, above models will always embed nodes at symmetric positions into same embedding vectors, which means that such nodes are indistinguishable from the GNN's point of view.

Heuristics that alleviate the above issues include assigning an unique identifier to each node (Kipf & Welling, 2017; Hamilton et al., 2017a) or using locally assigned node identifiers plus pre-trained transductive node features (Zhang & Chen, 2018). However, such models are not scalable and cannot generalize to unseen graphs where the canonical node ordering is not available. In contrast, P-GNNs can capture positional information without sacrificing other advantages of GNNs.

One alternative method to incorporate positional information is utilizing a graph kernel, which crucially rely on the positional information of nodes and inspired our P-GNN model. Graph kernels implicitly or explicitly map graphs to a Hilbert space. Weisfeiler-Lehman and Subgraph kernels have been incorporated into deep graph kernels (Yanardag & Vishwanathan, 2015) to capture structural properties of neighborhoods. Gärtner et al. (2003) and Kashima et al. (2003) also proposed graph kernels based on random walks,

[1] Code and data are available in https://github.com/JiaxuanYou/P-GNN

which count the number of walks two graphs have in common (Sugiyama & Borgwardt, 2015). Kernels based on shortest paths were first proposed in (Borgwardt & Kriegel, 2005).

3. Preliminaries

3.1. Notation and Problem Definition

A graph can be represented as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is the node set and $\mathcal{E}$ is the edge set. In many applications where nodes have attributes, we augment G with the node feature set $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ where $\mathbf{x}_i$ is the feature vector associated with node v_i.

Predictions on graphs are made by first embedding nodes into a low-dimensional space which is then fed into a classifier, potentially in an end-to-end fashion. Specifically, a node embedding model can be written as a function $f : \mathcal{V} \to \mathcal{Z}$ that maps nodes $\mathcal{V}$ to d-dimensional vectors $\mathcal{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_n\}, \mathbf{z}_i \in \mathbb{R}^d$.

3.2. Limitations of Structure-aware Embeddings

Our goal is to learn embeddings that capture the local network structure as well as retain the global network position of a given node. We call node embeddings to be *position-aware*, if the embedding of two nodes can be used to (approximately) recover their shortest path distance in the network. This property is crucial for many prediction tasks, such as link prediction and community detection. We show below that GNN-based embeddings cannot recover shortest path distances between nodes, which may lead to suboptimal performance in tasks where such information is needed.

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.*

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.*

For example, most graph neural networks compute node embeddings by aggregating information from each node's q-hop neighborhood, and are thus structure-aware. In contrast, (long) random-walk-based embeddings such as DeepWalk and Node2Vec are position-aware, since their objective function forces nodes that are close in the shortest path to also be close in the embedding space. In general, structure-aware embeddings cannot be mapped to position-aware embeddings. Therefore, when the learning task requires node positional information, only using structure-aware embeddings as input is not sufficient:

Proposition 1. *There exists a mapping g that maps structure-aware embeddings $f_{s_q}(v_i), \forall v_i \in \mathcal{V}$ to position-aware embeddings $f_p(v_i), \forall v_i \in \mathcal{V}$, if and only if no pair of nodes have isomorphic local q-hop neighbourhood graphs.*

Proposition 1 is proved in the Appendix. The proof is based on the identifiability arguments similar to the proof of Theorem 1 in (Hamilton et al., 2017a), and also explains why in some cases GNNs may perform well in tasks requiring positional information. However, in real-world graphs such as molecules and social networks, the structural equivalences between nodes' local neighbourhood graphs are quite common, making GNNs hard to identify different nodes. Furthermore, the mapping g essentially memorizes the shortest path distance between a pair of structure-aware node embeddings whose local neighbourhoods are unique. Therefore, even if the GNN perfectly learns the mapping g, it cannot generalize to the mapping to new graphs.

4. Proposed Approach

In this section, we first describe the P-GNN framework that extends GNNs to learn position-aware node embeddings. We follow by a discussion on our model designing choices. Last, we theoretically show how P-GNNs generalize existing GNNs and learn position-aware embeddings.

4.1. The Framework of P-GNNs

We propose Position-aware Graph Neural Networks that generalize the concepts of Graph Neural Networks with two key insights. First, when computing the node embedding, instead of only aggregating messages computed from a node's local network neighbourhood, we allow P-GNNs to *aggregate messages from anchor-sets*, which are randomly chosen subsets of all the nodes (Figure 2, left). Note that anchor sets get resampled every time the model is run forward. Secondly, when performing message aggregation, instead of letting each node aggregate information independently, the aggregation is *coupled across all the nodes* in order to distinguish nodes with different positions in the network (Figure 2, middle). We design P-GNNs such that each node embedding dimension corresponds to messages computed with respect to one anchor-set, which makes the computed node embeddings position-aware (Figure 2, right).

P-GNNs contain the following key components:

- k anchor-sets S_i of different sizes.
- Message computation function F that combines feature information of two nodes with their network distance.
- Matrix $\mathbf{M}$ of anchor-set messages, where each row i is an anchor-set message $\mathcal{M}_i$ computed by F.
- Trainable aggregation functions AGG_M, AGG_S that aggregate/transform feature information of the nodes in the anchor-set and then also aggregate it across the anchor-sets.

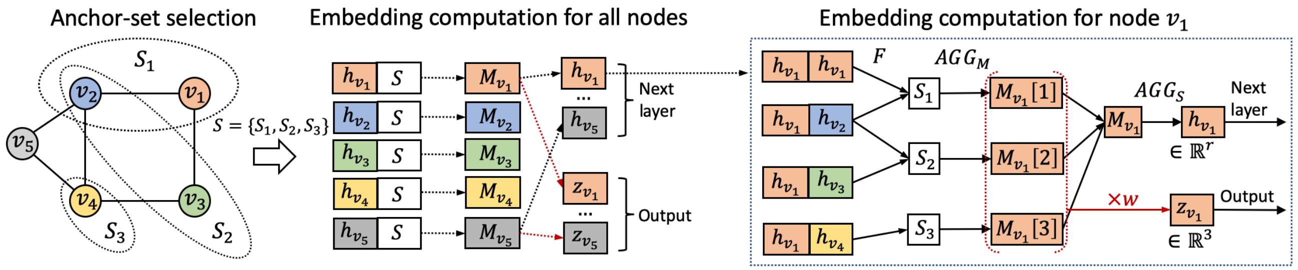

Figure 2. P-GNN architecture. P-GNN first samples multiple anchor-sets $S = \{S_1, S_2, S_3\}$ of different sizes (**Left**). Then, position-aware node embeddings $\mathbf{z}_{v_i}$ are computed via messages M_{v_i} between a given node v_i and the anchor-sets S_i which are shared across all the nodes (**Middle**). To compute the embedding $\mathbf{z}_{v_1}$ for node v_1, one layer of P-GNN first computes messages via function F and then aggregates them via a learnable function AGG_M over the nodes in each anchor-set S_i to obtain a matrix of anchor-set messages $\mathbf{M}_{v_1}$. The message matrix $\mathbf{M}_{v_1}$ is then further aggregated using a learnable function AGG_S to obtain node v_1's message $\mathbf{h}_{v_1}$ that can be passed to the next level of P-GNN. At the same time a learned vector $\mathbf{w}$ is used reduce $\mathbf{M}_{v_1}$ into a fixed-size position-aware embedding $\mathbf{z}_{v_1}$ which is the output of the P-GNN (**Right**).

- Trainable vector $\mathbf{w}$ that projects message matrix $\mathbf{M}$ to a lower-dimensional embedding space $\mathbf{z} \in \mathbb{R}^k$.

Algorithm 1 summarizes the general framework of P-GNNs. A P-GNN consists of multiple P-GNN layers. Concretely, the l^{th} P-GNN layer first samples k random anchor-sets S_i. Then, the i^{th} dimension of the output node embedding $\mathbf{z}_v$ represents messages computed with respect to the i^{th} anchor-set S_i. Each dimension of the embedding is obtained by first computing the message from each node in the anchor-set via message computation function F, then applying a message aggregation function AGG_M, and finally applying a non-linear transformation to get a scalar via weights $\mathbf{w} \in \mathbb{R}^r$ and non-linearity σ. Specifically, the message from each node includes distances that reveal node positions as well as feature-based information from input node features. The message aggregation functions are the same class of functions as used by existing GNNs. We further elaborate on the design choices in Section 4.3.

P-GNNs are position-aware. The output embeddings $\mathbf{z}_v$ are position-aware, as each dimension of the embedding encodes the necessary information to distinguish structurally equivalent nodes that reside in different parts of the graph. Note that if we permute the dimensions of all the node embeddings $\mathbf{z}_v$, the resulting embeddings are equivalent to the original embeddings because they carry the same node positional information with respect to (permuted order of) anchor-sets $\{S_i\}$.

Multiple P-GNN layers can be naturally stacked to achieve higher expressive power. Note that unlike GNNs, we cannot feed the output embeddings $\mathbf{z}_v$ from the previous layer to the next layer, because the dimensions of $\mathbf{z}_v$ can be arbitrarily permuted; therefore, applying a fixed non-linear transformation over this representation is problematic. The deeper reason we cannot feed $\mathbf{z}_v$ to the next layer is that the position of a node is always *relative* to the chosen anchor-sets; thus, canonical position-aware embeddings do not exist. Therefore, P-GNNs also compute structure-aware messages $\mathbf{h}_v$, which are computed via an order-invariant message aggregation function that aggregates messages *across anchor-sets*, and are then fed into the next P-GNN layer as input.

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 $\sigma$; Layer $l \in [1, L]$
**Output:** Position-aware embedding $\mathbf{z}_v$ for every node $v$
$\mathbf{h}_v \leftarrow \mathbf{x}_v$
for $l = 1, \dots, L$ do
  $S_i \sim \mathcal{V}$ for $i = 1, \dots, k$
  for $v \in \mathcal{V}$ do
    $\mathbf{M}_v = \mathbf{0} \in \mathbb{R}^{k \times r}$
    for $i = 1 \dots, k$ do
      $\mathcal{M}_i \leftarrow \{F(v, u, \mathbf{h}_v, \mathbf{h}_u), \forall u \in S_i\}$
      $\mathbf{M}_v[i] \leftarrow \text{AGG}_M(\mathcal{M}_i)$
    end for
    $\mathbf{z}_v \leftarrow \sigma(\mathbf{M}_v \cdot \mathbf{w})$
    $\mathbf{h}_v \leftarrow \text{AGG}_S(\{\mathbf{M}_v[i], \forall i \in [1, k]\})$
  end for
end for
$\mathbf{z}_v \in \mathbb{R}^k, \forall v \in \mathcal{V}$
```

4.2. Anchor-set Selection

We rely on Bourgain's Theorem to guide the choice of anchor-sets, such that the resulting representations are guaranteed to have low distortion. Specifically, distortion measures the faithfulness of the embeddings in preserving distances when mapping from one metric space to another metric space, which is defined as follows:

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 $\forall u, v \in \mathcal{V}$, $\frac{1}{\alpha} d(u, v) \le d'(f(u), f(v)) \le d(u, v)$.*

Theorem 1 states the Bourgain Theorem (Bourgain, 1985), which shows the existence of a low distortion embedding that maps from any metric space to the l_p metric space:

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 $O(\log n)$.*

A constructive proof of Theorem 1 (Linial et al., 1995) provides an algorithm to construct an $O(\log^2 n)$ dimensional embedding via anchor-sets, as summarized in Theorem 2:

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:*

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

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.

The proposed P-GNNs can be viewed as a generalization of the embedding method in Theorem 2, where the distance metric d is generalized via message computation function F and message aggregation function AGG_M that accounts for both node feature information and position-based similarities (Section 4.3). Using this analogy, Theorem 2 offers two insights for selecting anchor-sets in P-GNNs. First, $O(\log^2 n)$ anchor-sets are needed to guarantee low distortion embedding. Second, these anchor-sets have sizes distributed exponentially. Here, we illustrate the intuition behind selecting anchor-sets with different sizes via the 1-hop shortest path distance defined in Equation 2. Suppose that the model is computing embeddings for node v_i. We say an anchor-set *hits* node v_i if v_i or any of its one-hop neighbours is included in the anchor-set. Small anchor-sets can provide positional information with high certainty, because when a small anchor-set hits v_i, we know that v_i is located close to one of the very few nodes in the small anchor-set. However, the probability that such small anchor-set hits v_i is low, and the anchor-set is uninformative if it misses v_i. On the contrary, large anchor-sets have higher probability of hitting v_i, thus sampling large anchor-sets can result in high sample efficiency. However, knowing that a large anchor-set hits v_i provides little information about its position, since v_i might be close to any of the many nodes in the anchor-set. Therefore, choosing anchor-sets of different sizes balances the trade-off and leads to efficient embeddings.

Following the above principle, P-GNNs choose $k = c \log^2 n$ random anchor-sets, denoted as $S_{i,j} \subset \mathcal{V}$, where $i = 1, \dots, \log n, j = 1, \dots, c \log n$ and c is a hyperparameter. To sample an anchor-set $S_{i,j}$, we sample each node in $\mathcal{V}$ independently with probability $\frac{1}{2^i}$.

4.3. Design decisions for P-GNNs

In this section, we discuss the design choices of the two key components of P-GNNs: the message computation function F and the message aggregation functions AGG.

Message Computation Function F. Message computation function $F(v, u, \mathbf{h}_v, \mathbf{h}_u)$ has to account for both position-based similarities as well as feature information. Position-based similarities are the key to reveal a node's positional information, while feature information may include other side information that is useful for the prediction task.

Position-based similarities can be computed via the shortest path distance, or, for example, personalized PageRank (Jeh & Widom, 2003). However, since the computation of shortest path distances has a $O(|\mathcal{V}|^3)$ computational complexity, we propose the following q-hop shortest path distance

$$d^q_{sp}(v, u) = \begin{cases} d_{sp}(v, u), & \text{if } d_{sp}(v, u) \le q, \\ \infty, & \text{otherwise} \end{cases} \tag{2}$$

where d_{sp} is the shortest path distance between a pair of nodes. Note that 1-hop distance can be directly identified from the adjacency matrix, and thus no additional computation is needed. Since we aim to map nodes that are close in the network to similar embeddings, we further transform the distance $s(v, u) = \frac{1}{d^q_{sp}(v,u)+1}$ to map it to a $(0, 1)$ range.

Feature information can be incorporated into $\mathbf{h}_u$ by passing in the information from the neighbouring nodes, as in GCN (Kipf & Welling, 2017), or by concatenating node features $\mathbf{h}_v$ and $\mathbf{h}_u$, similar to GraphSAGE (Hamilton et al., 2017a), although other approaches like attention can be used as well (Velickovic et al., 2018). Combining position and feature information can then be achieved via concatenation or product. We find that simple product works well empirically. Specifically, we find the following message passing function F performs well empirically

$$F(v, u, \mathbf{h}_v, \mathbf{h}_u) = s(v, u)\text{CONCAT}(\mathbf{h}_v, \mathbf{h}_u) \tag{3}$$

Message Aggregation Functions AGG. Message aggregation functions aggregate information from a set of messages (vectors). Any permutation invariant function, such as MEAN, MIN, MAX, SUM, can be used, and non-linear transformations are often applied before and/or after the aggregation to achieve higher expressive power (Zaheer et al., 2017). We find that using simple MEAN aggregation function provides good results, thus we use it to instantiate both AGG_M and AGG_S.

5. Theoretical Analysis of P-GNNs

5.1. Connection to Existing GNNs

P-GNNs generalize existing GNN models. From P-GNN's point of view, existing GNNs use the same anchor-set message aggregation techniques, but use different anchor-set selection and sampling strategies, and only output the structure-aware embeddings $\mathbf{h}_v$.

GNNs either use deterministic or stochastic neighbourhood aggregation (Hamilton et al., 2017a). Deterministic GNNs can be expressed as special cases of P-GNNs that treat each individual node as an anchor-set and aggregate messages based on q-hop distance. In particular, the function F in Algorithm 1 corresponds to the message aggregation function of a deterministic GNN. In each layer, most GNNs aggregate information from a node's one-hop neighbourhood (Kipf & Welling, 2017; Velickovic et al., 2018), corresponding to using 1-hop distance to compute messages, or directly aggregating k-hop neighbourhood (Xu et al., 2018), corresponding to computing messages within k-hop distance. For example, a GCN (Kipf & Welling, 2017) can be written as choosing $\{S_i\} = \{v_i\}$, $\textsc{Agg}_M = \textsc{Mean}$, $\textsc{Agg}_S = \textsc{Mean}$, $F = \frac{1}{d^1_{sp}(v,u)+1}\mathbf{h}_u$, and the output embedding is $\mathbf{h}_u$ in the final layer.

Stochastic GNNs can be viewed as P-GNNs that sample size-1 anchor-sets, but each node's choice of anchor-sets is different. For example, GraphSAGE (Hamilton et al., 2017a) can be viewed as a special case of P-GNNs where each node samples k size-1 anchor-sets and then computes messages using 1-hop shortest path distance anchor-set, followed by aggregation $\textsc{Agg}_S$. This understanding reveals the connection between stochastic GNNs and P-GNNs. First, P-GNN uses larger anchor-sets thereby enabling higher sample efficiency (Sec 4.2). Second, anchor-sets that are shared across all nodes serve as reference points in the network, consequently, positional information of each node can be obtained from the shared anchor-sets.

5.2. Expressive Power of P-GNNs

Next, we show that P-GNNs provide a more *general class of inductive bias* for graph representation learning than GNNs; therefore, are more expressive to learn both structure-aware and position-aware node embeddings.

We motivate our idea by considering pairwise relation prediction between nodes. Suppose a pair of nodes u, v are labeled with label y, using labeling function $d_y(u, v)$, and our goal is to predict y for unseen node pairs. From the perspective of representation learning, we can solve the problem via learning an embedding function f that computes the node embedding $\mathbf{z}_v$, where the objective is to maximize the likelihood of the conditional distribution $p(y|\mathbf{z}_u, \mathbf{z}_v)$. Generally, an embedding function takes a given node v and the graph G as input and can be written as $\mathbf{z}_v = f(v, G)$, while $p(y|\mathbf{z}_u, \mathbf{z}_v)$ can be expressed as a function $d_z(\mathbf{z}_u, \mathbf{z}_v)$ in the embedding space.

As shown in Section 3.2, GNNs instantiate f via a function $f_\theta(v, S_v)$ that takes a node v and its q-hop neighbourhood graph S_v as arguments. Note that S_v is independent from S_u (the q-hop neighbourhood graph of node u) since knowing the neighbourhood graph structure of node v provides no information on the neighbourhood structure of node u. In contrast, P-GNNs assume a more general type of inductive bias, where f is instantiated via $f_\phi(v, S)$ that aggregates messages from random anchor-sets S that are shared across all the nodes, and nodes are differentiated based on their different distances to the anchor-sets S. Under this formulation, each node's embedding is computed similarly as in the stochastic GNN when combined with a proper q-hop distance computation (Section 5.1). However, since the anchor-sets S are shared across all nodes, pairs of node embeddings are correlated via anchor-sets S, and are thus no longer independent. This formulation implies a joint distribution $p(\mathbf{z}_u, \mathbf{z}_v)$ over node embeddings, where $\mathbf{z}_u = f_\phi(u, S)$ and $\mathbf{z}_v = f_\phi(v, S)$. In summary, *learning node representations* can be formalized with the following two types of objectives:

- GNN representation learning objective:

$$\begin{aligned}\min_\theta\ & \mathbb{E}_{u\sim V_{train}, v\sim V_{train}, S_u\sim p(V), S_v\sim p(V)} \\ & \mathcal{L}(d_z(f_\theta(u, S_u), f_\theta(v, S_v)) - d_y(u, v))\end{aligned} \tag{4}$$

- P-GNN representation learning objective:

$$\begin{aligned}\min_\theta\ & \mathbb{E}_{u\sim V_{train}, v\sim V_{train}, S\sim p(V)} \\ & \mathcal{L}(d_z(f_\phi(u, S), f_\phi(v, S)) - d_y(u, v))\end{aligned} \tag{5}$$

where d_y is the target similarity metric determined by the learning task, for example, indicating links between nodes or membership to the same community, and d_z is the similarity metric in the embedding space, usually the l_p norm.

Optimizing Equations 4 and 5 gives representations of nodes using joint and marginal distributions over node embeddings, respectively. If we treat u, v as random variables from G that can take values of any pair of nodes, then the mutual information between the joint distribution of node embeddings and any $Y = d_y(u, v)$ is larger than that between the marginal distributions and Y: $I(Y; X_{joint}) \geq I(Y; X_{marginal})$, where $X_{joint} = (f_\phi(u, S_u), f_\phi(v, S_v)) \sim p(f_\phi(u, S_u), f_\phi(v, S_v))$; $X_{marginal} = (f_\theta(u, S), f_\theta(v, S)) \sim p(f_\theta(u, S)) \otimes p(f_\theta(v, S))$, where $\otimes$ is the Kronecker product. The gap of this mutual information is great, if the target task $d_y(u, v)$ is related to the positional information of nodes which can be captured by the shared choice of anchor-sets. Thus, we conclude that P-GNNs, which embed nodes using the joint distribution of their distances to common anchors, have more expressive power than existing GNNs.

5.3. Complexity Analysis

In P-GNNs, every node communicates with $O(\log^2 n)$ anchor-sets in a graph with n nodes and e edges. Suppose on average each anchor-set contains m nodes, then there are $O(mn \log^2 n)$ message communications in total. In contrast, the number of communications is $O(ne)$ for existing GNNs. The complexity of P-GNN does not depend on the average degree of nodes in G, and thus P-GNN will be more efficient than a GNN of the graph is dense. In practice, we also observe that the computation can be sped up by using a simplified aggregation $\textsc{Agg}_S$, while only slightly sacrificing predictive performance. Here for each anchor-set, we only aggregate message from the node closest to a given node v. This will remove the factor m in the complexity of P-GNNs, making the complexity $O(n \log^2 n)$. We use this implementation in the experiments.

6. Experiments

6.1. Datasets

We perform experiments on both synthetic and real datasets. We use the following datasets for a link prediction task:

- **Grid** . 2D grid graph representing a 20× 20 grid with $|V| = 400$ and no node features.
- **Communities**. Connected caveman graph (Watts, 1999) with 1% edges randomly rewired, with 20 communities where each community has 20 nodes.
- **PPI**. 24 Protein-protein interaction networks (Zitnik & Leskovec, 2017). Each graph has 3000 nodes with avg. degree 28.8, each node has 50 dimensional feature vector.

We use the following datasets for pairwise node classification tasks which include community detection and role equivalence prediction[2].

- **Communities**. The same as above-mentioned community dataset, with each node labeled with the community it belongs to.
- **Emails**. 7 real-world email communication graphs from SNAP (Leskovec et al., 2007) with no node features. Each graph has 6 communities and each node is labeled with the community it belongs to.
- **Protein**. 1113 protein graphs from (Borgwardt et al., 2005). Each node is labeled with a functional role of the protein. Each node has a 29 dimensional feature vector.

6.2. Experimental setup

Next we evaluate P-GNN model on both transductive and inductive learning settings.

Transductive learning. In the transductive learning setting, the model is trained and tested on a given graph with a fixed node ordering and has to be re-trained whenever the node ordering is changed or a new graph is given. As a result, the model is allowed to augment node attributes with unique one-hot identifiers to differentiate different nodes. Specifically, we follow the experimental setting from (Zhang & Chen, 2018), and use two sets of 10% existing links and an equal number of nonexistent links as test and validation sets, with the remaining 80% existing links and equal number of nonexistent links used as the training set. We report the test set performance when the best performance on the validation set is achieved, and we report results over 10 runs with different random seeds and train/validation splits.

Inductive learning. We demonstrate the inductive learning performance of P-GNNs on pairwise node classification tasks for which it is possible to transfer the positional information to a new unseen graph. In particular, for inductive tasks, augmenting node attributes with one-hot identifiers restricts a model's generalization ability, because the model needs to generalize across scenarios where node identifiers can be arbitrarily permuted. Therefore, when the dataset does not come with node attributes, we only consider using constant order-invariant node attributes, such as a constant scalar, in our experiments. Original node attributes are used if they are available.

We follow the transductive learning setting to sample links, but only use order-invariant attributes. When multiple graphs are available, we use 80% of the graphs for training and the remaining graphs for testing. Note that we do not allow the model to observe ground-truth graphs at the training time. For the pairwise node classification task, we predict whether a pair of nodes belongs to the same community/class. In this case, a pair of nodes that do not belong to the same community are a negative example.

6.3. Baseline models

So far we have shown that P-GNNs are a family of models that differ from the existing GNN models. Therefore, we compare variants of P-GNNs against most popular GNN models. To make a fair comparison, all models are set to have similar number of parameters and are trained for the same number of epochs. We fix model configurations across all the experiments. (Implementational details are provide in the Appendix.) We show that even the simplest P-GNN models can significantly outperform GNN models in many tasks, and designing more expressive P-GNN models is an interesting venue for future work.

[2] Inductive position-aware node classification is not well-defined due to permutation of labels in different graphs. However pairwise node classification, which only decides if nodes are of the same class, is well defined in the inductive setting.

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	$\mathbf{0.808} \pm 0.003$

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

GNN variants. We consider 4 variants of GNNs, each with three layers, including GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017a), Graph Attention Networks (GAT) (Velickovic et al., 2018) and Graph Isomorphism Network (GIN) (Xu et al., 2019). Note that in the context of link prediction task, our implementation of GCN is equivalent to GAE (Kipf & Welling, 2016).

P-GNN variants. We consider 2 variants of P-GNNs, either with one layer or two layers (labeled 1L, 2L): (1) P-GNNs using truncated 2-hop shortest path distance (P-GNN-F); (2) P-GNNs using exact shortest path distance (P-GNN-E).

6.4. Results

Link prediction. In link prediction tasks two nodes are generally more likely to form a link, if they are close together in the graph. Therefore, the task can largely benefit from position-aware embeddings. Table 1 summarizes the performance of P-GNNs and GNNs on a link prediction task. We observe that P-GNNs significantly outperform GNNs across all datasets and variants of the link prediction taks (inductive vs. transductive). P-GNNs perform well in all inductive link prediction settings, for example improve AUC score by up to 66% over the best GNN model in the grid dataset. In the transductive setting, P-GNNs and GNNs achieve comparable performance. The explanation is that one-hot encodings of nodes help GNNs to memorize node IDs and differentiate symmetric nodes, but at the cost of expensive computation over $O(n)$ dimensional input features and the failure of generalization to unobserved graphs. On the other hand, P-GNNs can discriminate symmetric nodes by their different distances to anchor-sets, and thus adding one-hot features does not help their performance. In addition, we observe that when graphs come with rich features (e.g., PPI dataset), the performance gain of P-GNNs is smaller, because node features may already capture positional information. Quantifying how much of the positional information is already captured by the input node features is an interesting direction left for future work. Finally, we show that the "fast" variant of the P-GNN model (P-GNN-F) that truncates expensive shotest distance computation at 2 still achieves comparable results in many datasets.

Pairwise node classification. In pairwise node classification tasks, two nodes may belong to different communities but have similar neighbourhood structures, thus GNNs which focus on learning structure-aware embeddings will not perform well in this tasks. Table 2 summarizes the performance of P-GNNs and GNNs on pairwise node classification tasks. The capability of learning position-aware embeddings is crucial in the Communities dataset, where all P-GNN variants nearly perfectly detect memberships of nodes to communities, while the best GNN can only achieve 0.620 ROC AUC, which means that P-GNNs give 56% relative improvement in ROC AUC over GNNs on this task. Similar significant performance gains are also observed in Email and Protein datasets: 18% improvement in ROC AUC on Email and 39% improvement of P-GNN over GNN on Protein dataset.

7. Conclusion

We propose Position-aware Graph Neural Networks, a new class of Graph Neural Networks for computing node embeddings that incorporate node positional information, while retaining inductive capability and utilizing node features. We show that P-GNNs consistently outperform existing GNNs in a variety of tasks and datasets.

Acknowledgements

This research has been supported in part by Stanford Data Science Initiative, NSF, DARPA, Boeing, Huawei, JD.com, and Chan Zuckerberg Biohub.

References

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Belkin, M. and Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, pp. 585–591, 2002.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining*, pp. 74–81, 2005.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Bourgain, J. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2): 46–52, 1985.

Gärtner, T., Flach, P., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pp. 129–143. 2003.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *International Conference on Machine Learning*, 2017.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864. ACM, 2016.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017a.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 2017b.

Jeh, G. and Widom, J. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, pp. 271–279. Acm, 2003.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*, pp. 321–328, 2003.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

Leskovec, J., Kleinberg, J., and Faloutsos, C. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1 (1):2, 2007.

Linial, N., London, E., and Rabinovich, Y. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710. ACM, 2014.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Selsam, D., Lamm, M., Bunz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a sat solver from single-bit supervision. *International Conference on Learning Representations*, 2019.

Sugiyama, M. and Borgwardt, K. M. Halting in random walk kernels. In *Advances in Neural Information Processing Systems*, pp. 1639–1647, 2015.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *Graph attention networks*, 2018.

Watts, D. J. Networks, dynamics, and the small-world phenomenon. *American Journal of sociology*, 105(2): 493–527, 1999.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. *International Conference on Machine Learning*, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *International Conference on Learning Representations*, 2019.

Yanardag, P. and Vishwanathan, S. V. N. Deep graph kernels. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374, 2015.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018a.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4805–4815, 2018b.

You, J., Liu, B., Ying, R., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in Neural Information Processing Systems*, 2018a.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. Graphrnn: Generating realistic graphs with deep autoregressive models. In *International Conference on Machine Learning*, 2018b.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in Neural Information Processing Systems*, pp. 3391–3401, 2017.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 2018.

Zitnik, M. and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33 (14):i190–i198, 2017.