Directed Graph Auto-Encoders

Georgios Kollias, Vasileios Kalantzis, Tsuyoshi Idé, Aurélie Lozano, Naoki Abe

IBM Research
T. J. Watson Research Center
{gkollias, vkal, tide, aclozano, nabe}@us.ibm.com

Abstract

We introduce a new class of auto-encoders for directed graphs, motivated by a direct extension of the Weisfeiler-Leman algorithm to pairs of node labels. The proposed model learns pairs of interpretable latent representations for the nodes of directed graphs, and uses parameterized graph convolutional network (GCN) layers for its encoder and an asymmetric inner product decoder. Parameters in the encoder control the weighting of representations exchanged between neighboring nodes. We demonstrate the ability of the proposed model to learn meaningful latent embeddings and achieve superior performance on the directed link prediction task on several popular citation network datasets.

Introduction

Graph-structured data are ubiquitous, commonly encountered in diverse domains, ranging from biochemical interaction networks, to networks of social and economic transactions. A graph introduces dependencies between its connected nodes, thus algorithms designed to work solely with feature vectors of isolated nodes as inputs can yield suboptimal results. One way to remedy this issue, without reverting to more complex graph algorithms, is to enhance the representation of a graph node so that both its features and embedding graph structure are captured in a single vector.

Graph Convolutional Networks (GCNs) produce vectorial representations of nodes that are graph-aware and have been successfully used in downstream learning tasks including node classification, link prediction and graph classification. The construction of GCNs falls into two categories: spatial-based and spectral-based. Spatial-based GCNs are conveniently described as Message Passing Neural Networks (MPNNs) detailing the steps for aggregating information from neighbor graph nodes (Gilmer et al. 2017; Micheli 2009; Niepert, Ahmed, and Kutzkov 2016). They adopt a local view of the graph structure around each node and are straightforward to describe and lightweight to compute; however they also need local customizations to enhance the performance of the representations they produce for downstream learning tasks (Veličković et al. 2017). Spectral-based GCNs originate in graph signal processing perspectives (Bruna et al. 2014). They are based on the graph Laplacian, so they inherently adopt a global graph view (Defferrard, Bresson, and Vandergheynst 2016; Kipf and Welling 2016a). However they incur more computational cost, typically addressed by approximating their convolutional filter.

In this work we focus on GCN-based models for representing the nodes of directed graphs (encoding), so that we can faithfully reconstruct their directed edges (decoding). Our goal is to identify both whether two nodes u and v should be connected or not (which is the only goal for the undirected case) and whether their connection has the direction $u \mapsto v$ or $v \mapsto u$ or both. Many applications depend critically on this distinctionality. For example in (directed) citation graphs it cannot be the case that a publication cites a work that is published later in time, in (directed) causal graphs a causal node should prepend any of its effects, in knowledge graphs subject nodes are expected to point to object nodes. As a consequence, failing to identify the correct orientation even in a single edge could severely disrupt downstream tasks: time ordering can become contradicting, paths to root causes can be erroneously blocked, flow computations between entities can be totally wrong.

Our encoder follows a color refinement scheme for directed graphs that reduces to the standard Weisfeiler-Leman algorithm. Coupled with an unsymmetric decoder, our directed graph auto-encoder can accurately infer missing directed links from the limited, incomplete graph it has access to during training time.

Our contributions are two-fold. First, we propose a novel variant of the Weisfeiler-Leman algorithm that clearly emphasizes the dual role of directed graph nodes as both sources and targets of directed links. This abstracts, for the first time, the alternating update of authority and hub scalar values in HITS, and the relations between left and right singular vectors computed in SVD and the GCN-based approaches for directed graphs inspired by them. Second, we design parameterized GCN layers for updating the pair-of-vectors representation of the source and target roles of directed graph nodes in an alternating manner, and use these layers as the encoder in a directed graph auto-encoder architecture. We demonstrate that the parameterization we introduce is performance-critical for learning latent representations, and the proposed model can outperform state-of-the-art methods for the *directed* link prediction task on several popular citation network datasets

in terms of area under the ROC curve (AUC) and average precision (AP) metrics.

Preliminaries and background

We can generally describe the dependencies of a network as a directed graph $G(V, E, w)$, i.e., a weighted *dependency graph*. Here V is the set of $n = |V|$ graph nodes and $E = \{(i, j) \in V \times V : i \mapsto j\}$ is the set of its $m = |E|$ directed edges, expressed as node pairs. Finally $w : V \times V \to \mathbb{R}$ is the edge weight function, with $w(i, j)$ being a scalar capturing the "strength" of the dependency $i \mapsto j$ *iff* $(i, j) \in E$ - and vanishing otherwise. Following a linear algebra perspective, we will represent $G(V, E, w)$ as an $n \times n$ sparse, weighted, adjacency matrix $\mathbf{A}$. This matrix has m non-vanishing entries and its (i, j) entry is set equal to the respective weight $w(i, j)$, i.e., $\mathbf{A}[\mathtt{i,\ j}] = w(i, j)$. Throughout the rest of this paper, we use $\tilde{\mathcal{N}}^+(i) = \mathcal{N}^+(i) \cup \{i\}$ ($\tilde{\mathcal{N}}^-(i) = \mathcal{N}^-(i) \cup \{i\}$) and $\mathtt{deg}^+(i) = |\tilde{\mathcal{N}}^+(i)|$ ($\mathtt{deg}^-(i) = |\tilde{\mathcal{N}}^-(i)|$), to denote the neighbor node sets of the outgoing (incoming) edges and outgoing (incoming) degrees of a node i - including itself. Analogously, the corresponding diagonal matrices with the outdegrees (indegrees) along their diagonal will be denoted as $\tilde{\mathbf{D}}^+$ ($\tilde{\mathbf{D}}^-$), where we top with a tilde mark the names of adjacency matrices with added self-links. Note that for undirected, unweighted graphs, the corresponding adjacency matrices are symmetric and binary, i.e., $\tilde{\mathbf{D}}$ is a diagonal matrix where $\tilde{d}_{ii}$ is the original degree of node i increased by 1 (because of the added self-link).

Dual vector encoding of directed graph nodes

Consider a directed edge $i \mapsto j$, with weight $w(i, j)$, where i is the source node and j is the target node. Now, let's assume that node i is equipped with a pair of vectors in $\mathbb{R}^k$, $1 \leq i \leq n$: (i) vector $\mathbf{s}_i$ encodes i's role as a source, which is the same for any of the directed edges it participates as a source, and (ii) vector $\mathbf{t}_i$ encodes i's role as a target; similarly for node j. The similarity of nodes i and j in building the weighted directed edge $i \mapsto j$ could then be captured by a similarity function $\mathtt{sim}(\cdot, \cdot)$ which ideally evaluates to the true edge weight: $\mathtt{sim}(\mathbf{s}_i, \mathbf{t}_j) = w(i, j)$.

An immediate choice for the similarity function is the dot product: $\mathtt{sim}(\mathbf{s}_i, \mathbf{t}_j) = \mathbf{s}_i^\top \mathbf{t}_j$, with the encodings originally realized as column vectors. We then can compactly evaluate the weights of all edges, by buliding two matrices, $\mathbf{S}$ and $\mathbf{T}$, where $\mathbf{S}[\mathtt{i,\ :}] = \mathbf{s}_i^\top$ and $\mathbf{T}[\mathtt{j,\ :}] = \mathbf{t}_j^\top$, for all $1 \leq i, j \leq n$, and requiring $\mathbf{A} = \mathbf{S}\mathbf{T}^\top$. This particular choice has been explored in (Ou et al. 2016), which leverages the singular value decomposition ($\mathtt{SVD}$) $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ and sets $\mathbf{S} = \mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}$ and $\mathbf{T} = \mathbf{V}\mathbf{\Sigma}^{\frac{1}{2}}$. The authors also explore a variant based on truncated SVD.

Weisfeiler-Leman (WL) algorithm and connection to GCNs

One-dimensional Weisfeiler-Leman (1-WL) One dimensional Weisfeiler-Leman (1-WL) algorithm is a well-studied approach for assigning distinct labels to the nodes of undirected, unweighted graphs with different topological roles (Weisfeiler and Leman 1968). Given adjacency information in the form of neighborhood lists $\mathcal{N}(i) = [k : i \mapsto k \vee k \mapsto i]$, $\forall i \in [0, n)$, 1-dim WL iteratively updates a node's label by computing a bijective hash of its neigbors' labels, and mapping this to a unique label. The procedure terminates when the hash-and-map procedure stabilizes. A detailed sketch of the 1-WL algorithm is provided in the Appendix.

Graph Convolutional Network (GCN) Graph Convolutional Networks (GCNs) suggest convolution operators for graph signals defined over graph nodes. Following the early analysis in (Hammond, Vandergheynst, and Gribonval 2011) and the expansions in (Defferrard, Bresson, and Vandergheynst 2016), GCNs were particularly popularized by Kipf et al. (Kipf and Welling 2016a). In particular, the work in (Kipf and Welling 2016a) focused on undirected, unweighted graphs, such that the adjacency matrices are symmetric and binary. Each node i is initially assumed encoded by a $k = k_0$ dimensional vector $\mathbf{x}_i$, so all node encodings can be collected in an $n \times k$ matrix $\mathbf{X} = \mathbf{X}^{(0)}$. The goal is to *transform* the node embeddings so that a downstream task such as node classification is more accurate.

The proposed transformation contains a succession of *graph convolutional layers* of the form:

$$\mathbf{Z}^{(t)} \leftarrow \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^{(t)} \mathbf{W}^{(t)} \tag{1}$$

interspersed with *nonlinear layers*, like $\mathtt{ReLU}$ and $\mathtt{softmax}$. The quantity $\mathbf{W}^{(t)}$ is a *learnable* $k_t \times k_{t+1}$ matrix of weights for the t^{th} *graph convolutional layer* ($t = 0, 1, \ldots$). The algorithm in (Kipf and Welling 2016a) implements the transformation of the original encodings $\mathbf{X}$:

$$\mathbf{Z} \leftarrow \mathtt{softmax}(\hat{\mathbf{A}}\ \mathtt{ReLU}(\hat{\mathbf{A}}\ \mathbf{X}\ \mathbf{W}^{(0)})\ \mathbf{W}^{(1)}), \tag{2}$$

where $\hat{\mathbf{A}} := \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$.

Connecting 1-WL to GCNs In (Morris et al. 2019) the connection of 1-WL to 1-GNNs is explored. Their basic 1-GNN model assumes the form

$$f^{(t)}(v) = \sigma\left(f^{(t-1)}(v) \cdot W_1^{(t)} + \sum_{w \in N(v)} f^{(t-1)}(w) \cdot W_2^{(t)}\right) \tag{3}$$

In this, $f^{(t)}(v)$, which is the row feature vector of node v at layer $t > 0$, is computed by aggregating its feature vector and the feature vectors of its neighbors at the previous layer $t - 1$, after first multiplying them respectively by parameter matrices $W_1^{(t)}$ and $W_2^{(t)}$. It follows GCNs are 1-GNNs. These results establish the connection of 1-WL to 1-GNNs (3):

- **[Theorem 1 in (Morris et al. 2019)]** For all $t \geq 0$ and for all choices of weights $\mathbf{W}^{(t)} = (W_1^{(t')}, W_2^{(t')})_{t' \leq t}$, coloring $c_l^{(t)}$ refines encoding $f^{(t)}$: $c_l^{(t)} \sqsubseteq f^{(t)}$. This means that for any nodes u, w in G_{st}, $c_l^{(t)}(u) = c_l^{(t)}(w)$ implies $f^{(t)}(u) = f^{(t)}(w)$.
- **[Theorem 2 in (Morris et al. 2019)]** For all $t \geq 0$ there exists a sequence of $\mathbf{W}^{(t)}$ and a 1-GNN architecture such that the colorings and the encodings are equivalent: $c_l^{(t)} \equiv f^{(t)}$ (i.e. $c_l^{(t)} \sqsubseteq f^{(t)}$ and $f^{(t)} \sqsubseteq c_l^{(t)}$).

DiGAE: Directed Graph Auto-Encoder

Weisfeiler-Leman (WL) algorithm and connection to encoder layers in DiGAE

We now extend 1-WL for coloring pairs of node labels in directed graphs. We can formally prove that this extension reduces to standard 1-WL over a bipartite, undirected graph. This allows us to connect it to 1-GNNs with a special structure for the parameter matrices, which corresponds to the directed graph convolutional layers in the encoder module of our DiGAE architecture which we define next. For the case of shared weight matrices we can even remove this special structure requirement.

Coloring pairs of node labels in directed graphs To extend 1-WL for directed graphs, we equip each node of the graph with 2 labels, one for capturing its role as a "source" of directed edges emanating from it and another one for its role as a "target" node for directed edges pointing to it. Then at each step, we propose the "source" label of a node to be a bijective function of its current "source" label and the multiset of "target" labels of the nodes it points to. In parallel, its "target" label will be updated to the bijective mapping of its current target label and the "source" labels of the nodes pointing to it. For the detailed algorithm, please refer to the Appendix.

Neural Source and Target encodings The propagation of "source" and "target" node labels along their directed edges, as in the extension of the 1-WL algorithm, suggests a new graph convolutional network layer. Similarly to the way the GCN layer is closely connected to standard 1-WL, we propose a novel *directed graph convolutional layer* that transforms a pair of source and target encoding vectors for each node, in analogy to our extension for directed graphs. We assume that each node i is originally encoded by a pair of vectors, its *source* $\mathbf{s}_i^{(0)}$ and *target* $\mathbf{t}_i^{(0)}$ encodings, and collect these encodings as rows in matrices $\mathbf{S}^{(0)}$ and $\mathbf{T}^{(0)}$.

We can now define the t^{th} graph convolutional layer for updating the *source* encodings by aggregating the transformed and normalized *target* encodings in the neighborhood as:

$$\mathbf{S}^{(t+1)} \leftarrow \left(\tilde{\mathbf{D}}^{+}\right)^{-\beta} \tilde{\mathbf{A}} \left(\tilde{\mathbf{D}}^{-}\right)^{-\alpha} \mathbf{T}^{(t)} \mathbf{W}_T^{(t)} \tag{4}$$

In this work we propose tunable parameters α and β for weighting the degrees in message passing. This subtle modification can have important effects in performance for the link prediction task as demonstrated in the experimental section.

Similarly we update the *target* encodings by aggregating the transformed and normalized *source* encodings in the neighborhood:

$$\mathbf{T}^{(t+1)} \leftarrow \left(\tilde{\mathbf{D}}^{-}\right)^{-\alpha} \tilde{\mathbf{A}}^{\top} \left(\tilde{\mathbf{D}}^{+}\right)^{-\beta} \mathbf{S}^{(t)} \mathbf{W}_S^{(t)} \tag{5}$$

where $\mathbf{W}_T^{(t)}$ and $\mathbf{W}_S^{(t)}$ are the (learnable) linear tranformations for target and encoding encodings prior to their propagation. Defining $\hat{\mathbf{A}} = \left(\tilde{\mathbf{D}}^{+}\right)^{-\beta} \tilde{\mathbf{A}} \left(\tilde{\mathbf{D}}^{-}\right)^{-\alpha}$, we can compactly express our proposed graph convolutional layer by the following pair of tranformations:

$$\begin{aligned} \mathbf{S}^{(t+1)} &\leftarrow \hat{\mathbf{A}}\, \mathbf{T}^{(t)}\, \mathbf{W}_T^{(t)} \\ \mathbf{T}^{(t+1)} &\leftarrow \hat{\mathbf{A}}^{\top}\, \mathbf{S}^{(t)}\, \mathbf{W}_S^{(t)}. \end{aligned} \tag{6}$$

Reduction to 1-WL We consider the transformation of our directed, unweighted graph $G(V, E)$ to its *bipartite representation* $G_{st}(V_s, V_t, E_{st})$ with $V_s = V = [0, n)$, $V_t = [n, 2\ n)$ and $E_{st} = \{\{i, j+n\} | (i, j) \in E\}$ (Bang-Jensen and Gutin 2008). The bipartite graph G_{st} is assumed undirected and it follows that for the set $\mathcal{N}_{st}(v)$ of the immediate neighbors of any of its nodes $v \in V(G_{st}) = V_s \cup V_t$:

$$\mathcal{N}_{st}(v) = \begin{cases} \{j+n | (v, j) \in E\} & \text{if } 0 \le v < n \\ \{i | (i, v-n) \in E\} & \text{if } n \le v < 2 \end{cases} \tag{7}$$

We also assume that the $2n$ nodes of G_{st} are initially colored with two colors in the set $\{s, t\}$ according to a coloring function function $l : V_s \cup V_t \to \Sigma$ with $l(v) = s$, if $0 \le v < n$ and $l(v) = t$, if $n \le v < 2\ n$. In other words (G_{st}, l) is a labeled graph, so we can directly apply the standard 1-WL iterative algorithm for color refinement. In each iteration $t \ge 0$, 1-WL computes a node coloring $c_l^{(t)} : V(G_{st}) \to \Sigma$ with an arbitrary codomain of colors Σ. We initialize this sequence of colorings as $c_l^{(0)} = l$ and at $t > 0$, node coloring is updated as:

$$c_l^{(t)}(v) = \texttt{HASH}\left((c_l^{(t-1)}(v), \{\{c_l^{(t-1)}(w) | w \in \mathcal{N}_{st}(v)\}\}\right)$$

Termination of this standard 1-WL algorithm is guaranteed after at most $|V(G_{st})| = 2\ n$ iterations and reached when $|c_l^{(t)}| = |c_l^{(t-1)}|$ for some k.

Essentially, the bipartite representation maps a node $i \in V$ in the directed graph G to a pair of nodes $(i \in V_s, i+n \in V_t)$ in the undirected, bipartite graph G_{st}: $i \in V_s$ represents $i \in V$ as a *source* of directed edges in G and $i+n \in V_t$ represents $i \in V$ as a *target* of directed edges in G. In addition, identical color pair labels (s, t), initially assigned to all nodes $i \in V$ (in the directed graph G) are mapped to: (i) an initial label s for node $i \in V_s$ and (ii) an initial label t for node $i + n \in V_t$ (in the undirected, bipartite graph G_{st}). So in our refinement scheme for a node i in G, it holds $(c_{l,s}^{(0)}(i), c_{l,t}^{(0)}(i)) = (s, t)$ while for its corresponding nodes $i, i+n$ in G_{st}, $c_l^{(0)}(i) = s$ and $c_l^{(0)}(i+n) = t$.

We can now prove that the color pairs computed at each step k and for each node i by our refinement scheme on a directed graph G can equivalently be computed by standard 1-WL on the undirected, bipartite G_{st} by pairing the colors of its nodes i and $i + n$. In other words:

Theorem 1.

$$c_{l,s}^{(t)}(i) = c_l^{(t)}(i), c_{l,t}^{(t)}(i) = c_l^{(t)}(i+n) \tag{8}$$

for all layers $t \ge 0$ and $i \in [0, n)$.

Proof. We use induction. For $t = 0$ this holds by our initialization convention: $c_{l,s}^{(0)}(i) = c_l^{(0)}(i) = s$ and $c_{l,t}^{(0)}(i) = c_l^{(0)}(i+n) = t$.

Let us now assume that this also holds for some t and find out for $t+1$. Consider a node with index $i \in [0, n)$. Its neighbors $w = j+n \in \mathcal{N}_{st}(i)$ in G_{st} map to the outlink-neighbors $j \in \mathcal{N}^+(i)$ in G according to our bipartite representation (see Equation (7). These neighbors $w = j + n \in \mathcal{N}_{st}(i)$ have color labels $c_l^{(t)}(j+n)$ which according to (8) are equal to $c_{l,t}^{(t)}(i)$:

$$c_l^{(t+1)}(i) = \texttt{HASH}((c_l^{(t)}(i), \{\{c_l^{(t)}(w)|w \in \mathcal{N}_{st}(i)\}\}) = \texttt{HASH}((c_{l,s}^{(t)}(i), \{\{c_{l,t}^{(t)}(j)|j \in \mathcal{N}^+(i)\}\}) \quad (9)$$

Similarly for a node with index $i + n \in [n, 2n)$, its neighbors $w = j \in \mathcal{N}_{st}(i)$ in G_{st} map to the inlink-neighbors $j \in \mathcal{N}^-(i-n)$ in G according to Equation 7. The node with index $i + n$ has color label equal to $c_{l,t}^{(t)}$ and the inlink neighbors j, since $j \in [0, n)$ have color labels equal to $c_{l,s}^{(t)}$ (see 8):

$$c_l^{(t+1)}(i+n) = \texttt{HASH}((c_l^{(t)}(i+1), \{\{c_l^{(t)}(w)|w \in \mathcal{N}_{st}(i)\}\}) = \texttt{HASH}((c_{l,t}^{(t)}(i), \{\{c_{l,s}^{(t)}(j)|j \in \mathcal{N}^-(i)\}\}) \quad (10)$$

According to our extension to 1-WL, the refinement we propose reads: $c_{l,s}^{(t+1)}(i) = \texttt{HASH}\left((c_{l,s}^{(t)}(i), \{\{c_{l,t}^{(t)}(j)|j \in \mathcal{N}^+(i)\}\}\right)$ and $c_{l,t}^{(t+1)}(i) = \texttt{HASH}\left((c_{l,t}^{(t)}(i), \{\{c_{l,s}^{(t)}(j)|j \in \mathcal{N}^-(i)\}\}\right)$. By comparing with equations (9) and (10) it follows that $c_{l,s}^{(t+1)} = c_l^{(t+1)}(i)$ and $c_{l,t}^{(t+1)} = c_l^{(t+1)}(i+n)$ which completes the proof. □

Connecting 1-WL to the encoder layers in DiGAE We can encode the initial labels $c_l^{(0)}(v)$ of the nodes in our bipartite representation G_{st} by considering two arbitrary, nonequal, vectors $\mathbf{s}$, $\mathbf{t}$ in $\mathbb{R}^{e\times 1}$ and setting $f^{(0)}(v) = \mathbf{s}^\top$ for nodes $v \in [0, n)$ and $f^{(0)}(v) = \mathbf{t}^\top$ for nodes $v \in [n, 2n)$; e can be as small as 1 (i.e. for one-hot encoding). These encodings are consistent with the initial labels (i.e. they are different for nodes with different initial colors). Given the consistent encodings and our reduction of our color refinement to 1-WL for G_{st}, two results readily apply, from theorems 1 and 2 in (Morris et al. 2019). These results readily establish the connection of our 1-WL to 1-GNNs (3).

Alternatively, we can encode the initial labels $c_l^{(0)}(v)$ of the nodes in our bipartite representation G_{st} by vectors $f^{(0)}(v)$ as follows. For nodes $v \in [0, n)$ we assume the encoding $f^{(0)}(v) = [\mathbf{s}^\top, \mathbf{0}]$ and for nodes $v \in [n, 2n)$ we set $f^{(0)}(v) = [\mathbf{0}, \mathbf{t}^\top]$ where $\mathbf{0} \in \mathbb{R}^{1\times e}$ denotes the zero row vector. Our encodings are in $\mathbb{R}^{1\times 2e}$ and they are also consistent with our initial labels. In this encoding scheme, the first e elements (last e elements) of the initial feature vector $f^{(0)}(v)$ are non-vanishing for nodes in G_{st} capturing the source (target) role of nodes in G.

We can ensure this property is preserved for feature vectors $f^{(t)}(v), t > 0$ under the iteration in (3) by additionally selecting block-diagonal matrices for $W_1^{(t)}$ and block-antidiagonal matrices for $W_2^{(t)}$:

$$W_1^{(t)} = \begin{pmatrix} W_{1S}^{(t)} & O \\ O & W_{1T}^{(t)} \end{pmatrix}, W_2^{(t)} = \begin{pmatrix} O & W_{2S}^{(t)} \\ W_{2T}^{(t)} & O \end{pmatrix} \quad (11)$$

where $W_{1S}^{(t)}, W_{1T}^{(t)}, W_{2S}^{(t)}, W_{2T}^{(t)} \in \mathbb{R}^{e\times e}$ and $O = O^{e\times e}$ is the zero matrix.

This is straightforward to verify. For a node in $i \in [0, n)$ for which $f^{(t-1)}(i) = [\mathbf{s}_i^{(t-1)}, \mathbf{0}]$ its neighbors $j \in [n, 2n)$ will have encodings of the form $f^{(t-1)}(v) = [\mathbf{0}, \mathbf{t}_j^{(t-1)}]$. Substitution in Equation (3) yields:

$$f^{(t)}(i) = [\sigma({\mathbf{s}_i^{(t-1)}}^\top \cdot W_{1S}^{(t)} + \sum_{j\in N(i)} {\mathbf{t}_j^{(t-1)}}^\top \cdot W_{2T}^{(t)}), \mathbf{0}^{1\times e}]$$

so $f^{(t)}(i)$ still has only its *first* e elements non-vanishing, the same as $f^{(t-1)}(i)$. Similarly for $i \in [n, 2n)$ we get: $f^{(t)}(i) = [\sigma(\mathbf{0}^{1\times e}, {\mathbf{t}_i^{(t-1)}}^\top \cdot W_{1T}^{(t)} + \sum_{j\in N(i)} {\mathbf{s}_j^{(t-1)}}^\top \cdot W_{2S}^{(t)})]$, so $f^{(t)}(i)$ still has only its *last* e elements non-vanishing, the same as $f^{(t-1)}(i)$.

The motivation behind the selection of block-structured parameter matrices $W_1^{(t)}$ and $W_2^{(t)}$, is for allowing the extra flexibility of learning different sets of parameters for nodes with source and target roles (in undirected G_{st}). By removing self-links and setting $W_{1S}^{(t)} = W_{1T}^{(t)} = O^{e\times e}$, these vectors $f^{(t)}(i)$ can be seen to map directly to rows of matrices $\mathbf{S}^{(t)}$ and $\mathbf{T}^{(t)}$ in Equation (6). Alternatively, we can consider general (non-structured) $W_1^{(t)}$ and $W_2^{(t)}$ in our 1-GNN which corresponds to sharing learnt parameters for neural source and target encodings.

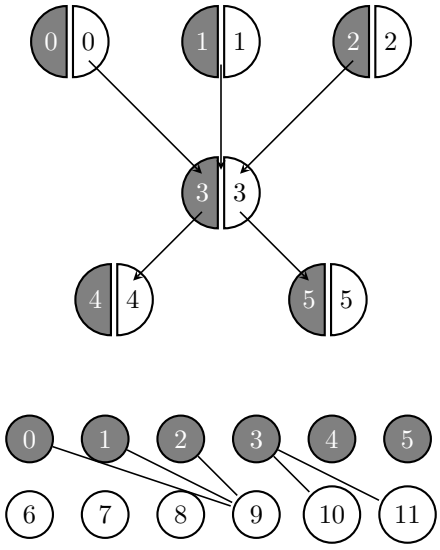

Figure 1: An example directed graph G with $n = 6$ nodes (top) and its bipartite representation G_{st} (bottom). Both graphs are colored with each node in G carrying a pair of source and target colors - respectively in left and right semidiscs. The example node 3 in G maps to nodes 3 and 9 in G_{st}.

Directed Graph AutoEncoder (DiGAE) model

In analogy to the Graph AutoEncoder (GAE) model in (Kipf and Welling 2016b), we can define its directed variant (DiGAE) by stacking two directed graph convolutional layers connected with `ReLU` nonlinearity for its *encoder* and a `sigmoid` applied to the inner product of the source and target encodings for its *decoder*. In particular, the *encoder* reads

$$\begin{aligned} \mathbf{Z}_S &= \hat{\mathbf{A}}\texttt{ReLU}(\hat{\mathbf{A}}^\top\ \mathbf{S}^{(0)}\ \mathbf{W}_S^{(0)})\mathbf{W}_T^{(1)} \\ \mathbf{Z}_T &= \hat{\mathbf{A}}^\top\texttt{ReLU}(\hat{\mathbf{A}}\ \mathbf{T}^{(0)}\ \mathbf{W}_T^{(0)})\mathbf{W}_S^{(1)} \end{aligned} \tag{12}$$

while the decoder for computing the reconstructed adjacency matrix $\bar{\mathbf{A}}$ is

$$\bar{\mathbf{A}} = \sigma(\mathbf{Z}_S\ \mathbf{Z}_T^\top) \tag{13}$$

In the sequel, we experiment with the single-layer DiGAE model, referred to as *DiGAE-1L*. This autoencoder has the same decoder as DiGAE and its encoder implements the pair of transformations $\mathbf{Z}_S = \mathbf{S}^{(1)} = \hat{\mathbf{A}}\mathbf{T}^{(0)}\mathbf{W}_T^{(0)}$ and $\mathbf{Z}_T = \mathbf{T}^{(1)} = \hat{\mathbf{A}}^\top\mathbf{S}^{(0)}\mathbf{W}_S^{(0)}$.

Related work

Ma et al (Ma et al. 2019) formally extend convolution to directed graphs by defining a normalized, symmetric directed Laplacian and leveraging the Perron vector of the induced transition probability matrix for weighing messaged passing. In (Tong et al. 2020a), Tong et al approximate the digraph Laplacian by means of Personalized PageRank, and this relaxed definition offers performance benefits and the ability to process directed graphs that are not necessarily strongly connected. The authors of Fast Directed GCN in (Li et al. 2020) also approximate the digraph Laplacian, by assuming the trivial Perron vector for regular graphs. In (Monti, Otness, and Bronstein 2018), a polynomial of selected motif Laplacians is used to filter node representations. Most notably, all aforementioned spectral-based extensions to convolution in directed graphs, produce single-vector representations, so the dual nature of a node as both a source and target of directed edges is not captured; we do not require the separate computation of the Perron vector and a scheme for weighing messages is automatically learnt. For convolution in directed knowledge graphs, we refer to (Kampffmeyer et al. 2019; Schlichtkrull et al. 2018).

In (Tong et al. 2020b), first and second order proximity kernels for directed graphs are combined to produce single-vector representations for graph nodes. Second order proximity kernels normalize the products of $\mathbf{A}$ and its transpose, and produce dual-vector intermediate representations, similarly to the work proposed in this paper. However, normalization of the products is graph-agnostic, the intermediate sparse matrix products they employ are computationally costly with potentiallly dense matrix outputs, the first order proximity kernels is symmetric. High-Order Proximity preserved Embeddings (HOPE) in (Ou et al. 2016) rely on matrix factorization (SVD) of a higher order proximity matrix while Asymmetric Proximity Preserving (APP) embeddings in (Zhou et al. 2017) rely on random walks with restart. These encoders are not GCN-based which is the focus in our work. Dual-vector representation can be enforced artificially: in (Salha et al. 2019), source/target GAE and VGAE models are based on graph auto-encoders for undirected graphs from the seminal work of Kipf and Welling (Kipf and Welling 2016b), where the single-vector representation (of even size) is assumed to be the concatenation of two-identically sized parts The gravity-inspired directed GCN architecture in (Salha et al. 2019) on the other hand produces single vector encodings using fixed normalization for the messages and embeds asymmetry in only one of its entries by fusing the importance of the target node and the distance of the node encodings at the ends of the directed edge.

The directed graph can be of a special kind. Message passing in GCNs for Directed Acyclic Graphs (DAGs) is explored in (Thost and Chen 2021). Or the directed edges can carry labels of different types in which case the undirected GCN can be extended to include terms denoting the additional aggregation of edge features (Jaume et al. 2019), in close analogy to the extension of WL test to a directed graph with edge labels (Orsini, Frasconi, and De Raedt 2015; Grohe et al. 2017). In another interesting view, the directed graph can first be converted to an undirected bipartite graph as in (Zhou, Hofmann, and Schölkopf 2005) driven by the source-target node duality.

Experiments

In this section we demonstrate the performance of the proposed approach on the *directed* link prediction task associated with two different datasets: (a) namely CoraML (2,995 nodes, 8,416 edges, 2,879 features), and (b) CiteSeer (3,312 nodes, 4,715 edges, 3,703 features). The CoraML dataset contains machine learning publications grouped into seven classes. The CiteSeer dataset contains scientific papers grouped into six classes. Each paper in CoraML and CiteSeer is represented by a one-hot vector indicating the presence or absence of a word from a dictionary.

We employ grid search for hyperparameter tuning: learning rate $\eta \in \{0.005, 0.01\}$, hidden layer dimension $d \in \{32, 64\}$ with $d/2$ for the latent space dimension, $(\alpha, \beta) \in \{0.0, 0.2, 0.4, 0.6, 0.8\}^2$ for DiGAE models and parameter $\lambda \in \{0.1, 1, 0, 10.0\}$ for Gravity GAE. For the final models we select hyperparameter values that maximize mean AUC computed on the validation set. In all cases, models are trained for 200 epochs, using Adam optimizer, without dropout, performing full-batch gradient descent.

We use Python and especially the PyTorch library and PyTorch Geometric (Fey and Lenssen 2019), which is a geometric deep learning extension library. We ran experiments for all models (ours and baselines) on a system equipped with an Intel(R) Core(TM) i7-8850H CPU @2.60GHz (6 cores/12 threads) and 32 GB of DDR4 memory @2400 MHz.

We consider the following directed link prediction task adapting the description in (Kipf and Welling 2016b) to digraphs.

Task: Directed link prediction We randomly remove 15% of the directed edges from the graph and train models on the

remaining edge set. Two thirds of the removed edges (i.e. 10% of all input graph edges) are used as actual edges for testing, one third of them (i.e. 5% of all input graph edges) as actual edges for validation. These test and validation sets also include the same number of fake directed edges (negative samples), as their actual ones: negative samples are generated by randomly connecting pairs of nodes which are not wired in the input graph. Validation sets are only used for hyperparameter tuning.

Results and Discussion

We compare the performance of our DiGAE models to Standard GAE in (Kipf and Welling 2016b), Source/Target GAE and Gravity GAE both in (Salha et al. 2019). Similarly to our models, these baselines are GCN-based. We use the TensorFlow implementations of baseline models from the authors of (Salha et al. 2019).[1]

We run a series of twenty experiments for each graph and selected model combination, and report the mean and standard deviation of the AUC and AP metrics, as well as their respective timings. Random train and test graph dataset splits are used for each such experiment and metrics are averaged over the set, to account for their sensitivity to the choice of the split, as empirically observed in (Shchur et al. 2018).

Results with node features supplied in all cases (feature-based configurations), are summarized in Table 1. We mark in bold the largest entry (or largest entries in the case of their overlapping intervals).

Mean AUC values for DiGAE-1L in particular consistently outperform other baselines for the citation graphs and the margin can be significant: in CiteSeer this is of the order of 6% for Gravity GAE and up to 18% compared to Standard GAE. For mean AP, margins are similarly in the 4% to 19% range, or up to 3% for CoraML. Varying (α, β) pair values has significant impact on the reconstruction metrics. As an example, the mean AUC can be as low as 73.73% and 86.10% for DiGAE-1L respectively for CoraML and CiteSeer for the selected η and d but for suboptimal (α, β) within the search grid. In the Appendix we include metrics tables for the full (α, β) grid collected during hyperparameter tuning. In experiments with citation graphs, DiGAE-1L is markedly faster by factors in the range $\times 5$ to $\times 15$. This is partly due to the different implementations, the fact that this is a single-layer only and for CiteSeer in particular to the smaller size for the output vector from hyperparameter tuning (16 vs 32 for a hidden encoding of 64 entries for baselines). Gravity GAE is the slowest, mainly due to the complexity of its decoder (pairwise distance computation).

The authors in (Salha et al. 2019) originally used the baselines with one-hot encoding of the nodes (feature-less configurations). For citation networks, node features capture similarity which is symmetric and this could conceptually hinder the identification of directionality in predicted links as in our task, which is an inherently asymmetric relation. For this reason, we also tested feature-less configurations for the baselines and interestingly got comparable results to the feature-based case (Table 2): our DiGAE-1L models from (Table 1) are top performers for both CoraML and CiteSeer datasets. Experiments with datasets from the WebKB collection and Pubmed are included in the Appendix.

[1] https://github.com/deezer/gravity_graph_autoencoders

Table 1: General directed link prediction for the *citation* graphs (feature-based configurations).

Dataset	Model	AUC	AP	Time (secs)
CoraML	DiGAE (ours)	88.10 +/- 1.70	89.83 +/- 1.39	9.56 +/- 0.48
CoraML	DiGAE 1-Layer (ours)	**94.09 +/- 0.66**	**94.10 +/- 0.77**	7.64 +/- 0.21
CoraML	Gravity GAE	92.35 +/- 0.57	**94.17 +/- 0.53**	51.56 +/- 0.13
CoraML	Source/Target GAE	91.66 +/- 0.52	92.60 +/- 0.47	36.88 +/- 0.20
CoraML	Standard GAE	89.54 +/- 1.14	91.40 +/- 1.11	36.55 +/- 0.21
CiteSeer	DiGAE (ours)	92.05 +/- 1.06	92.29 +/- 0.97	11.10 +/- 0.22
CiteSeer	DiGAE 1-Layer (ours)	**92.76 +/- 0.87**	**92.57 +/- 1.08**	4.12 +/- 0.11
CiteSeer	Gravity GAE	86.79 +/- 0.98	88.60 +/- 1.05	106.44 +/- 0.16
CiteSeer	Source/Target GAE	82.90 +/- 1.79	84.70 +/- 1.47	73.46 +/- 0.21
CiteSeer	Standard GAE	74.35 +/- 1.77	80.96 +/- 1.26	73.84 +/- 0.48

Table 2: General directed link prediction for the *citation* graphs (feature-less configurations).

Dataset	Model	AUC	AP
CoraML	Gravity GAE	91.56 +/- 0.71	93.62 +/- 0.59
CoraML	Source/Target GAE	91.85 +/- 0.58	92.91 +/- 0.64
CoraML	Standard GAE	89.42 +/- 0.84	92.09 +/- 0.64
CiteSeer	Gravity GAE	87.05 +/- 0.99	88.88 +/- 0.96
CiteSeer	Source/Target GAE	82.39 +/- 1.56	84.38 +/- 1.57
CiteSeer	Standard GAE	73.81 +/- 1.37	80.52 +/- 1.13

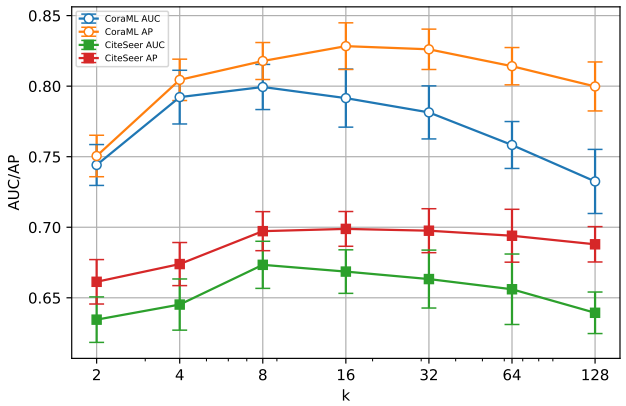

Figure 2: Truncated SVD based directed link prediction: AUC and AP metrics for CoraML and CiteSeer.

Truncated SVD baseline We also explore the performance of an approach based on truncated SVD for our directed link prediction task. For a given train/validation/test split, we use the directed edges that are available to use for training for building a *partial* adjacency matrix A_p of the underlying directed graph $G(V, E)$ and compute its truncated SVD, retaining the singular triplets corresponding to its largest k singular values, $\mathbf{A}_p \approx \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\top$. Then we consider the matrices for the source and target encodings of nodes $\mathbf{Z}_S = \mathbf{U}_k \mathbf{\Sigma}_k^{1/2}$

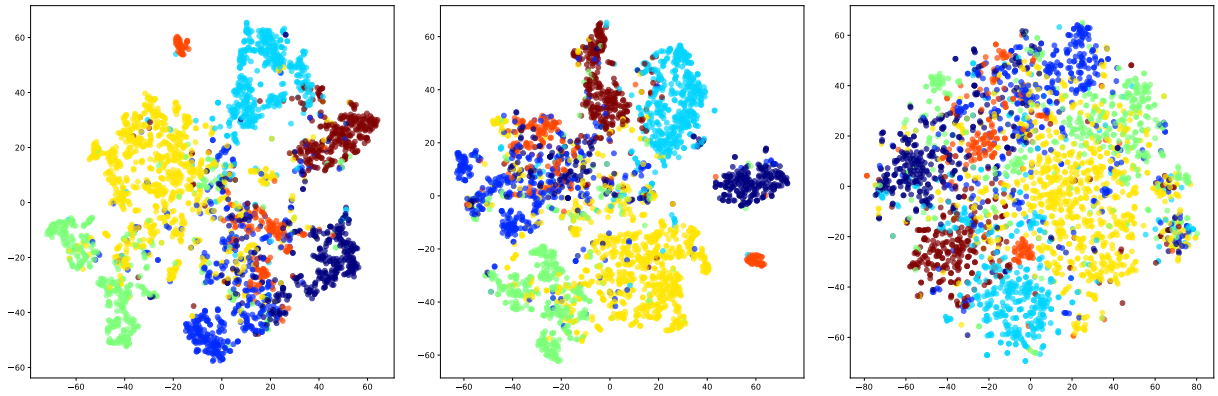

Figure 3: Left and center: t-SNE embeddings on source and target representations computed with our DiGAE-1L for CoraML. Right: embeddings of orginal feature vectors (colors indicate classes).

and $\mathbf{Z}_T = \mathbf{V}_k \mathbf{\Sigma}_k^{1/2}$ and use the asymmetric decoder from Equation (13) for predicting directed links. We use the same evaluation pipeline as in our GNN experiments and compute respective AUC and AP performance metrics for the graph reconstruction. For truncated SVD we experimented with implementations based on `ARPACK` (Lehoucq, Sorensen, and Yang 1998) and randomized SVD (Halko, Martinsson, and Tropp 2011) for $k \in \{2^i | i = 1, 2, \ldots, 7\}$ and we got very similar metrics for our input graphs (please refer to the Appendix for metrics tables). We repeated for 20 random graph splits for each graph, k and implementation combination. Figure 2 summarizes the results for the *largest* mean AUC and AP value for each graph, k combination. We observe that for 16-dimensional encoding vectors we get best results in reconstruction in terms of mean AP: 82.83% for CoraML and 69.88% for CiteSeer. These are significantly lower than 94.10% for CoraML and 92.57% for CiteSeer that the "neural" source and target encodings our DiGAE-1L produces. SVD encodes only connectivity while DiGAE models, being GNNs, leverage both connectivity *and* node features. The truncated SVD baseline is essentially the HOPE idea (Ou et al. 2016) with the proximity matrix being the adjacency matrix $\mathbf{A}$. For comparison and for the standard Katz proximity in HOPE, $(\mathbf{I} - \beta\mathbf{A})^{-1}\beta\mathbf{A}$, we computed *largest* mean AP values in reconstruction over $k \in \{2^i | i = 1, 2, \ldots, 7\}$, $\beta = 0.02$, applying 20 random graph splits for each k. We get 83.87% for CoraML and 67.29% for CiteSeer: comparable to truncated SVD baseline results and significantly lower than the respective DiGAE-1L metrics.

Clustering source and target encodings We used t-Distributed Stochastic Neighbor Embedding (t-SNE) (Van der Maaten and Hinton 2008) for reducing the dimension of source and target representations of nodes produced by DiGAE in order to visualize them. Figure 3 illustrates a crisp separation of points with different class labels and this is true for both source and target vectors. This separation is not present in their feature space. This implies that DiGAE embeddings are promising inputs for clustering purposes. The fact that the clusters can be independently identified in both the source and target space opens up the possibility for exploring sets of points that share cluster identity in both spaces, as the core ones for a label.

Vector hub and authority scores Source and target representations are expected to be the (vector) surrogates respectively of (scalar) hub and authority scores. Table 3 confirms this intuition. A source vector of particularly large magnitude for a node a will be more probable to yield large inner products with target vectors of other nodes b in the decoder. This means that a directed edge $a \mapsto b$ will be likely to appear, which will increase a's outdegree. Similarly, a large magnitude for a node's target vector encourages other nodes to connect to it and increase its indegree. In turn, outdegrees are known to correlate to hub scores and indegrees to authority and PageRank scores.

Table 3: Pearson correlation coefficients between the magnitudes of source and target vector encodings, and a collection of centrality scores (Hub/Authority and PageRank) and degrees (in/out), for all nodes in CoraML. Encodings were computed with DiGAE-1L.

	source magnitude	target magnitude
hub	**0.37**	0.06
outdegree	**0.82**	0.12
authority	0.05	**0.41**
pagerank	-0.01	**0.48**
indegree	0.08	**0.77**

Conclusions and future work

In this paper we present DiGAE, a new class of directed graph autoencoders that computes a pair of vector representations for each node. It exploits the asymmetry in input and output node degrees and further skews this by allowing exponents in scaling the features to enter as parameters. DiGAE outperforms state-of-the art GCN-based graph autoencoders on the directed link prediction task and can be an order of magnitude times faster in learning representations for CoraML and CiteSeer datasets. In future work, we plan to explore encoders that integrate scaling decisions that are *local* to each node.

References

Bang-Jensen, J.; and Gutin, G. Z. 2008. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 3844–3852.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 1263–1272. PMLR.

Grohe, M.; Kersting, K.; Mladenov, M.; and Schweitzer, P. 2017. Color refinement and its applications. *Van den Broeck, G.; Kersting, K.; Natarajan, S*, 30.

Halko, N.; Martinsson, P.-G.; and Tropp, J. A. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2): 217–288.

Hammond, D. K.; Vandergheynst, P.; and Gribonval, R. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2): 129–150.

Jaume, G.; Nguyen, A.-p.; Martínez, M. R.; Thiran, J.-P.; and Gabrani, M. 2019. edGNN: a Simple and Powerful GNN for Directed Labeled Graphs. *arXiv preprint arXiv:1904.08745*.

Kampffmeyer, M.; Chen, Y.; Liang, X.; Wang, H.; Zhang, Y.; and Xing, E. P. 2019. Rethinking knowledge graph propagation for zero-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 11487–11496.

Kipf, T. N.; and Welling, M. 2016a. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

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

Lehoucq, R. B.; Sorensen, D. C.; and Yang, C. 1998. *ARPACK users' guide - solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.* Software, environments, tools. SIAM. ISBN 978-0-89871-407-4.

Li, C.; Qin, X.; Xu, X.; Yang, D.; and Wei, G. 2020. Scalable graph convolutional networks with fast localized spectral filter for directed graphs. *IEEE Access*, 8: 105634–105644.

Ma, Y.; Hao, J.; Yang, Y.; Li, H.; Jin, J.; and Chen, G. 2019. Spectral-based graph convolutional network for directed graphs. *arXiv preprint arXiv:1907.08990*.

Micheli, A. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3): 498–511.

Monti, F.; Otness, K.; and Bronstein, M. M. 2018. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, 225–228. IEEE.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4602–4609.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*, 2014–2023. PMLR.

Orsini, F.; Frasconi, P.; and De Raedt, L. 2015. Graph invariant kernels. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence*, volume 2015, 3756–3762. IJCAI-INT JOINT CONF ARTIF INTELL.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 1105–1114.

Salha, G.; Limnios, S.; Hennequin, R.; Tran, V.-A.; and Vazirgiannis, M. 2019. Gravity-inspired graph autoencoders for directed link prediction. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 589–598.

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*, 593–607. Springer.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Thost, V.; and Chen, J. 2021. Directed Acyclic Graph Neural Networks. *arXiv preprint arXiv:2101.07965*.

Tong, Z.; Liang, Y.; Sun, C.; Li, X.; Rosenblum, D.; and Lim, A. 2020a. Digraph Inception Convolutional Networks. *Advances in Neural Information Processing Systems*, 33.

Tong, Z.; Liang, Y.; Sun, C.; Rosenblum, D. S.; and Lim, A. 2020b. Directed graph convolutional network. *arXiv preprint arXiv:2004.13970*.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2017. Graph Attention Networks. *6th International Conference on Learning Representations*.

Weisfeiler, B.; and Leman, A. 1968. The reduction of a graph to canonical form and the algebra which appears therein.

Zhou, C.; Liu, Y.; Liu, X.; Liu, Z.; and Gao, J. 2017. Scalable graph embedding for asymmetric proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Zhou, D.; Hofmann, T.; and Schölkopf, B. 2005. Semi-supervised learning on directed graphs. In *Advances in neural information processing systems*, 1633–1640.