# Subgraph Pattern Neural Networks for High-Order Graph Evolution Prediction

**Changping Meng, S Chandra Mouli, Bruno Ribeiro, Jennifer Neville**
Department of Computer Science
Purdue University
West Lafayette, IN
{meng40, chandr}@purdue.edu, {ribeiro, neville}@cs.purdue.edu

## Abstract

In this work we generalize traditional node/link prediction tasks in dynamic heterogeneous networks, to consider joint prediction over larger $k$-node induced subgraphs. Our key insight is to incorporate the unavoidable dependencies in the training observations of induced subgraphs into both the input features and the model architecture itself via high-order dependencies. The strength of the representation is its invariance to isomorphisms and varying local neighborhood sizes, while still being able to take node/edge labels into account, and facilitating inductive reasoning (i.e., generalization to unseen portions of the network). Empirical results show that our proposed method significantly outperforms other state-of-the-art methods designed for static and/or single node/link prediction tasks. In addition, we show that our method is scalable and learns interpretable parameters.

## Introduction

Learning predictive models of heterogeneous relational and network data is a fundamental task in machine learning and data mining (Getoor and Mihalkova 2011; Lao and Cohen 2010; Lin et al. 2015; Grover and Leskovec 2016; Nickel, Rosasco, and Poggio 2016). Much of the work in heterogeneous networks (graphs with node and edge labels) has focused on developing methods for label prediction or single link prediction. There has been relatively little development in methods that make joint predictions over larger substructures (e.g., induced $k$-node subgraphs). Recent research has shown rich higher-order organization of such networks (Benson, Gleich, and Leskovec 2016; Xu, Wickramarathne, and Chawla 2016) and complex subgraph evolution patterns within larger graphs (Paranjape, Benson, and Leskovec 2017). Applications range from predicting group activity on social networks (e.g., online social network ad revenues rely heavily on user activity), computational social science (e.g., predicting the dynamics of groups and their social relationships), relational learning (e.g., find missing and predicting future joint relationships in knowledge graphs).

The main challenge in learning a model to predict the evolution of labeled *subgraphs* is to jointly account for the induced subgraph dependencies that emerge from subgraphs sharing edges. Unlike node and edge prediction tasks, it is not clear how to describe an approximate model that can account for these dependencies. A variety of recent methods have developed heuristics to encode joint label and structure information into low dimensional node or edge embeddings, but it is unclear how these ad-hoc methods can properly address the induced subgraph dependencies (Lao and Cohen 2010; Nickel, Tresp, and Kriegel 2011; Dong et al. 2014; Lin et al. 2015; Grover and Leskovec 2016; Atwood and Towsley 2016; Nickel, Rosasco, and Poggio 2016; Rahman and Al Hasan 2016). Our empirical results show that these methods tend to perform poorly in induced subgraph prediction tasks.

The task of predicting induced subgraph evolution requires an approach that can take into account higher-order dependencies between the induced subgraphs (due to their shared edges and non-edges[1]). Our two main contributions are:
(1) We target the evolution of larger graph structures than nodes and edges, which, to the best of our knowledge, has never been focused before. Traditional link prediction tasks are simpler special cases of our task.
(2) We incorporate the unavoidable dependencies within the training observations of induced subgraphs into both the input features and the model architecture itself via high-order dependencies. We denote our model architecture a Subgraph Pattern Neural Network (SPNN) and show that its strength is due to a representation that is invariant to isomorphisms and varying local neighborhood sizes, can also take node/edge labels into account, and which facilitates inductive reasoning.

SPNN is a discriminative feedforward neural network with hidden layers that represent the *dependent* subgraph patterns observed in the training data. The input features of SPNN extend the definition of induced isomorphism density (Lovász and Szegedy 2006) to a local graph neighborhood in a way that accounts for joint edges and non-edges in the induced subgraphs. Moreover, SPNN is inductive (it can be applied to unseen portions of the graph), and is isomorphic-invariant, such the learned model is invariant to node permutations. We also show that SPNN learns to predict using an interpretable neural network structure.

[1] A non-edge marks the absence of an edge

## Heterogeneous Subgraph Prediction

In what follows we define the heterogeneous pattern prediction task and present a classification approach that uses a neural network classifier whose structure is based on *connected* induced subgraphs. In what follows, to avoid confusion with work on "learning low dimensional embeddings," we *avoid* using the correct-graph theoretic term *graph embeddings* (Borgs et al. 2008) in favor of the less standard term *induced subgraphs* of a smaller graph pattern into a larger graph.

**Data Definitions.** We consider a simple *heterogeneous* graph sequence $(G_n)_{n=1}^3$, where each graph $G_n = (V, E_n, \Phi_n, \Psi_n)$ is simple (i.e., without loops or multiple edges) and heterogeneous (i.e., with labeled (typed) nodes/edges). We denote the node and edge set of $G_n$ by $V(G_n)$ and $E(G_n)$, respectively. Node and edge labels of $G_n$ are defined by functions $\Phi_n$ and $\Psi_n$, respectively, s.t. $\Phi_n : V \to 2^{|A|}$, for a set of node classes $A$, and $\Psi_n : E \to 2^{|R|}$, for a set of edge types $R$. Clearly, $G_n$ can also represent directed graphs by adding direction labels over its edges.

**Definition 1** (**Induced Labeled Subgraphs**).
*Let $F$ and $G$ be two arbitrary heterogeneous graphs such that $|V(F)| \leq |V(G)|$. An induced subgraph of $F$ into $G$ is an adjacency preserving injective map $\gamma_F : V(F) \to V(G)$ s.t. for all pairs of vertices $i, j \in V(F)$, the pair $(\gamma_F(i), \gamma_F(j)) \in E(G)$ iff $(i, j) \in E(F)$, and all the corresponding node and edge labels of $i$ and $j$ match, i.e., $\Phi(i) = \Phi(\gamma_F(i))$, $\Phi(j) = \Phi(\gamma_F(j))$, and, if $(i, j) \in E(F) \implies \Psi((i, j)) = \Psi((\gamma_F(i), \gamma_F(j)))$.*

In the remainder of the paper, we consider these "$F$"s as small $k$-node graphs and refer to them as *subgraph patterns*.

**Definition 2** (**Task Definition**).
**Subgraph Patterns of Interest:** *The $k$-node subgraph patterns of interest are $\mathcal{F}^k = \{F_1, \ldots, F_c\}$, where $c \geq 1$, $|V(F_i)| = k, \forall i$.*
**Labels:** *In order to simplify the classification task, we further partition these patterns into sets with $r$ distinct "classes", which we denote $\mathcal{Y}_1^k, \ldots, \mathcal{Y}_r^k$ (as shown in Figure 1a).*
**Training data:** *$\mathcal{T}_1^k$ and $\mathcal{T}_2^k$ are the set of all $k$-node induced subgraphs of patterns $\mathcal{F}^k$ in $G_1$ and $G_2$, respectively, as described in Definition 1. For each induced subgraph $U \in \mathcal{T}_1^k$, we define its label $y_2(U)$ by looking at the pattern these same nodes form in $\mathcal{T}_2^k$, where $y_2(U) = r$, if the nodes $V(U)$ form an induced subgraph with pattern $F \in \mathcal{Y}_r^k$. Note that the patterns in $\mathcal{F}^k$ must encompass all possible evolution of the induced subgraphs in $\mathcal{T}_1^k$. The training data is*

$$\mathcal{D}_{train} = \{(U, y_2(U)) : U \in \mathcal{T}_1^k\}.$$

*Examples (Figure 1a, best seen in color): The induced subgraph $U \in \mathcal{T}_1^3$ shown in the blue oval, with vertices $V(U) = \{V_2, T_3, A_2\}$ (a venue, a topic, an author), has pattern $F =$ $\in \mathcal{F}^3$. The label of $U$ is $y_2(U) = 1$ as the vertices $V(U)$ form pattern $F =$ $\in \mathcal{Y}_1^3$ in $G_2$. The induced subgraph $U' \in \mathcal{T}_1^3$ shown in the red oval, $V(U') = \{V_1, T_1, A_1\}$, has pattern in $G_1$ and pattern in $G_2$, thus, $y_2(U') = 2$.*

**Prediction Task:** *Given the induced subgraphs in $\mathcal{T}_2^k$, our goal is to predict their corresponding pattern in $G_3$. These predicted patterns must be in $\mathcal{F}^k$.*

Traditional link prediction tasks (Liben-Nowell and Kleinberg 2007) can be seen as special instances of the task in Definition 2, where $k = 2$ and the target set of patterns $\mathcal{Y}_1^2$ consist of edges (i.e., 2-node connected induced subgraphs) and non-edges $\mathcal{Y}_2^2$. In the single link prediction case, the focus is on predicting individual links such as friendship links in Facebook, citation links in DBLP, or links in knowledge bases such as WordNet.

**Obtaining Training Data from Large Networks.** Let $\mathcal{T}_t^k$, be all $k$-node induced subgraphs with patterns $\mathcal{F}^k$ over $G_t$. Our training data consists of $\mathcal{T}_1^k$ and the future patterns of these induced subgraphs in $\mathcal{T}_2^k$, both which can be very large even for moderately small networks. We reduce *computational resources* needed to generate the **training data** by filtering the data of $\mathcal{T}_1^k$ as follows.

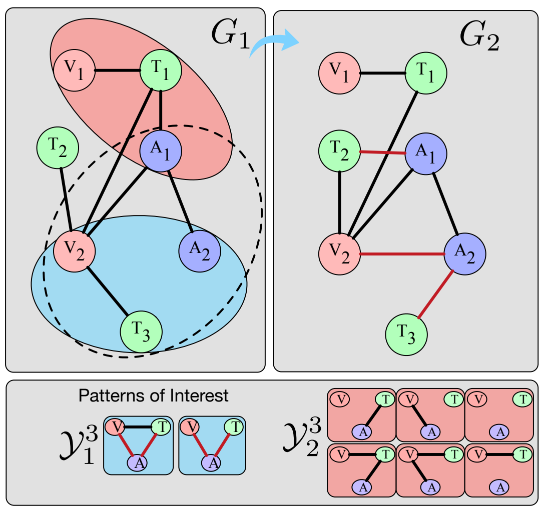
We construct a training dataset $\widetilde{\mathcal{T}}_1^k \subseteq \mathcal{T}_1^k$ such that a $k$-node induced subgraph $U \in \widetilde{\mathcal{T}}_1^k$ must belong to a larger $(k + \delta)$-node **connected** induced subgraph in $G_1$, $\delta \geq 1$. This constraint facilitates the identification of more *relevant* disconnected subgraphs of size $k$ without having to fully enumerate all the possibilities. By *relevant*, we mean that those $k$-node disconnected subgraphs are overwhelmingly more likely to evolve into connected patterns because the $k$ nodes have shortest paths of length up to $(k + \delta - 1)$ hops in $G_1$. Thus, the choice of $\delta$ is not arbitrary: we choose $\delta$ such that most of training examples with the labels we are most interested in predicting (e.g., Class 1 in Figure 1a) are still in $\widetilde{\mathcal{T}}_1^k$.

This filtering procedure also helps us quickly sample the training data from $G_1$ using a fast connected subgraph sampling method with known sampling bias (such that the bias can be removed) (Wang et al. 2014).

## (SPNN) Subgraph-Pattern Neural Network

Subgraph-Pattern Neural Network (SPNN) is our proposed classifier. SPNN is a 3-layer gated neural network with a sparse structure generated from the training data in a preprocessing step. The second neural network layer, which we call the Pattern Layer, is interpretable as it represents the $(k+\delta)$-node patterns in $G_1$ that were found while collecting the training data, described next. The neural network also has gates to deactivate the backpropagation of errors to the hidden units as we will describe later.

**Pattern Layer.** In the example of Figure 1a, the 3-node training example of induced subgraph $U$ in $G_1$, $V(U) = \{A_2, V_2, T_3\}$ (blue oval), belongs to a connected 4-node subgraph ($\{A_1, A_2, V_2, T_3\}$, the dotted oval) that matches the



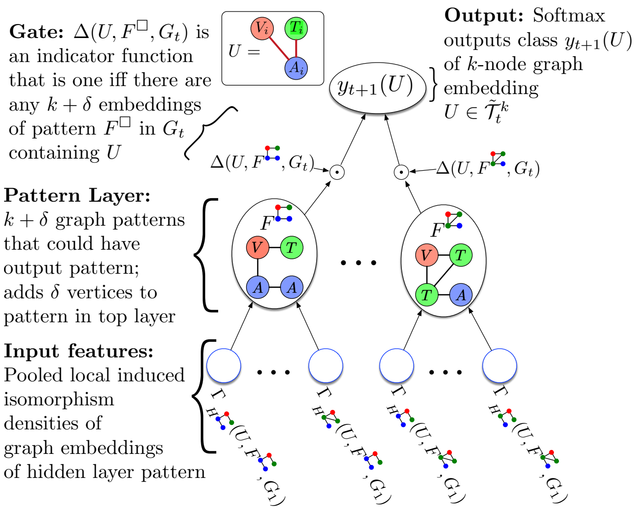

**Gate:** $\Delta(U, F^\square, G_t)$ is an indicator function that is one iff there are any $k+\delta$ embeddings of pattern $F^\square$ in $G_t$ containing $U$
$U =$
$V_i$
$T_i$
$A_i$
**Output:** Softmax outputs class $y_{t+1}(U)$ of $k$-node graph embedding $U \in \widetilde{\mathcal{T}}_t^k$
$y_{t+1}(U)$
$\Delta(U, F, G_t)$
$\Delta(U, F, G_t)$
**Pattern Layer:** $k+\delta$ graph patterns that could have output pattern; adds $\delta$ vertices to pattern in top layer
$F$
V
T
A
**Input features:** Pooled local induced isomorphism densities of graph embeddings of hidden layer pattern
$\Gamma_H(U, F, G_t)$
(b)

Figure 1: (a) Illustration of the training in a citation network with (A)uthors, (T)opics, (V)enues. At the top is the graph evolution $G_1$ to $G_2$, whose induced subgraphs are used as training data to predict the evolution of subgraphs in $G_2$ to $G_3$; below $\mathcal{Y}^3$ shows 3-node subgraph patterns partitioned into two classes. (b) Illustration of our Subgraph-Pattern Neural Network (SPNN) with training example $U$.

pattern $F$ represented in Figure 1b. More generally, the set of all such patterns is

$$\mathcal{F}_{t=1}^{\square(k+\delta)}(\mathcal{T}_{t=1}^{k}) = \{F^\square : \forall F^\square \in \mathcal{P}_{(conn)}^{(k+\delta)} \text{ s.t. } \exists U \in \mathcal{T}_1^k, \\ \exists R \in \text{Ind}(F^\square, G_1), \text{ and } R \in T_1^{(k+\delta)}(U, F^\square)\}, \tag{1}$$

where the square $\square$ indicates a connected subgraph pattern (e.g., , $\ldots$, ), $\mathcal{P}_{(conn)}^{(k+\delta)}$ is the set of **all** $(k+\delta)-$node **connected** graph patterns containing all possible node and edge labels, $\text{Ind}(F, G)$ denotes the set of induced subgraphs of $F$ into a graph $G$, and for $U \in \widetilde{\mathcal{T}}_1^k$ we define

$$T_1^{(k+\delta)}(U, F^\square) = \{(k+\delta)\text{-node induced connected} \\ \text{subgraphs of } F^\square \text{ at } G_1 \text{ having all nodes of U}\}. \tag{2}$$

For instance, $T_1^{(k+\delta)}(U, F)$ with $V(U) = \{V_2, A_2, T_3\}$ in Figure 1b. In practice, *we also mark the nodes of* $U \in \widetilde{\mathcal{T}}_1^k$ *that appear in the* $(k+\delta)$*-node patterns with unique special types, so we can distinguish their structural role in the larger* $(k+\delta)$*-node subgraph.*

Figure 1b illustrates the SPNN architecture using the task illustrated in Figure 1a as an example. For instance, we want to jointly predict whether an author $A$ will publish at a venue $V$ and in topic $T$ at $G_2$ given such author did not publish at venue $V$ or topic $T$ at $G_1$.

**Pattern Layer & Gates.** The hidden layer of SPNN represents $\mathcal{F}^{\square(k+\delta)} := \{F^{\square_1}, F^{\square_2}, \ldots\}$, **all observed** $(k+\delta)$-node patterns in the training data $\widetilde{\mathcal{T}}_1^k$. This procedure *only eliminates patterns that are not observed in the training data.* For example, in the illustration of Figure 1a, $\delta = 1$, there would be no 4-node patterns of a fully connected graph in $\mathcal{F}^{\square(3+1)}$ as there are no fully connected 4-node graphs in $G_1$.

For the training example $U \in \widetilde{\mathcal{T}}_t^k$, it may be the case that pattern $F^\square \in \mathcal{F}_t^{\square(k+\delta)}$ has no induced subgraph on $G_t$ that contains the example $U$, i.e., $T_t^{(k+\delta)}(U, F^\square) = \emptyset$. If this happens, we should not backpropagate the error of the hidden unit associated with $F^\square$. For instance, for $\delta = 1$ in the illustration of Figure 1a, the training data induced subgraph $U \in \widetilde{\mathcal{T}}_t^3$ with vertices $V(U) = \{A_1, T_1, V_1\}$ will only backpropagate the error to the hidden units matching the patterns of induced subgraphs $\{A_1, T_1, V_1, V_2\}$ and $\{A_1, T_1, V_1, A_2\}$. We use a gate function

$$\Delta(U, F^\square, G_t) = \mathbb{1}\{T_t^{(k+\delta)}(U, F^\square) \neq \emptyset\}, \tag{3}$$

with $T_t^{(k+\delta)}$ as defined in Eq.(2). The gate $\Delta(U, F^\square, G_t)$ ensures we are only training the neural network unit of $F^\square$ when the induced subgraph example $U$ applies to that unit.

Our pattern layer has an interpretable definition: each pattern neuron represents a larger subgraph pattern containing the target subgraph. If a specific neuron has a significant impact activating the output, we know that its corresponding pattern is important in the predictions.

**Input Features.** In what follows we define the features given to the input layers of SPNN. Our features need the definition of a *local* induced isomorphism density around the induced subgraph of pattern $F^\square$ on $G_t$, with $F^\square \in \mathcal{F}_t^{\square(k+\delta)}$.

**Definition 3** (Local induced isomorphism density). *Let $R$ be a induced subgraph of $G$ and let $F$ be a subgraph pattern s.t. $|V(G)| > |V(F)| \geq |V(R)|$. The local induced isomorphism density, $t_{local}$, rooted at $R$ with subgraph pattern $F$ is the proportion of induced subgraphs of $F$ at $G$ in a ball of radius $d$ from the nodes of $V(R)$. More precisely, $t_{local}(R, F, G, d) \propto |\text{LocInd}(R, F, G, d)|$, where* $\text{LocInd}(R, F, G, d) = \{R' \in \text{Ind}(F, G) : |V(R') \cup V(R)| - |V(R') \cap V(R)| \leq d\}$.

The quantity $t_{\text{local}}$ is the proportion of induced subgraphs of pattern $F$ at $G$ constrained to the set of vertices that are up to $d$ hops away from the set of nodes $V(R)$. If $G$ has a small diameter, $d$ should be small.

We now use $t_{\text{local}}$ to define the input features for an example $U \in \widetilde{\mathcal{T}}_t^k$. For each $F^\square \in \mathcal{F}_t^{\square(k+\delta)}$, there will be a vector $\phi(U, F^\square, G_t)$ of dimension $m_{F^\square}$ (to be defined below), where

$$(\phi(U, F^\square, G_t))_i := \Gamma_{H_i}(U, F^\square, G_t)$$
$$= \sum_{R \in T_t^{(k+\delta)}(U, F^\square)} t_{\text{local}}(R, H_i, G_t, d),\ H_i \in \mathcal{P}_{(conn)}^{(k+\delta)}, \quad (4)$$

where, as before, $\mathcal{P}_{(conn)}^{(k+\delta)}$ is the set of all possible $(k+\delta)$-node connected patterns. Each input feature $\Gamma_H$ is a pooled value of $t_{\text{local}}$ that counts the density of induced subgraphs of a $(k+\delta)$-node pattern $H$ around a ball of radius $d$ from the vertices $V(R)$, where $R$ is a $(k+\delta)$-node connected induced subgraph that contains the example $U$. Thus, $\Gamma_H$ sums the densities of induced subgraphs that *can have* up to $d+\delta$ nodes different from $U$. We only include $\Gamma_H$ in the vector $\phi(U, F^\square, G_t)$ if $\exists U \in \widetilde{\mathcal{T}}_t^k$ s.t. $\Gamma_H(U, F^\square, G_t) > 0$. As $m_{F^\square}$ is the number of non-zero values of $\Gamma$, then $m_{F^\square} \leq |\mathcal{P}_{(conn)}^{(k+\delta)}|$.

To illustrate the $\Gamma$ metric, consider pattern $F$ illustrated in Figure 1b and the training example $U$ as the induced subgraph $\{A_2, V_2, T_3\}$ in $G_1$ in Figure 1a. $U$ is contained in the connected 4-node subgraph with $V(R) = \{A_2, V_2, T_3, A_1\}$. The pattern $H$ has $\Gamma_{H}(U, F, G_t) = 1/4$ as there is only one induced subgraph $\{(T_2, V_2), (V_2, A_1), (A_1, T_1), (T_1, V_2)\}$ with pattern $H$ out of the 4 induced 4-node subgraphs that are within a radius of $d = 1$ of the nodes $V(R)$.

**The SPNN Classifier.** We now put all the different components together for a $r$-class classification task. Consider the class $y_{t+1}(U)$ as a one-of-$K$ encoding vector. For a $k$-node induced subgraph $U$ of $G_t$, the probability nodes $V(U)$ form an induced subgraph in $G_{t+1}$ with a pattern of class $i$, for $1 \leq i \leq r$, is

$$p(y_{t+1}(U); \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)})_i$$
$$= \text{softmax}((\mathbf{W}^{(1)} h_t(U; \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) + \mathbf{b}^{(1)})_i)\,,$$

where $\mathbf{b}^{(1)} \in \mathbb{R}^d$ is the bias of the output layer and $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times |\mathcal{F}_t^{\square(k+\delta)}|}$ are the linear weights of the pattern layer. The input to the pattern layer is

$$h_t(U; \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = (\Delta(U, F_1^\square, G_t) \cdot \sigma(\mathbf{b}_1^{(2)}$$
$$+ (\mathbf{W}_1^{(2)})^\intercal \phi(U, F_1^\square, G_t)), \Delta(U, F_2^\square, G_t) \cdot \sigma(\mathbf{b}_2^{(2)}$$
$$+ (\mathbf{W}_2^{(2)})^\intercal \phi(U, F_2^\square, G_t)), \ldots)\,,$$

where for each unit associated with $F_j^\square$, $j = 1, 2, \ldots$, we have $\mathbf{b}_j^{(2)} \in \mathbb{R}$ as the bias and $\mathbf{W}_j^{(2)}$ as the classifier weights, and $\sigma$ is an activation function (our empirical results use $\tanh$), the feature vector $\phi(U, F_j^\square, G_t)$ is as defined in Eq.(4), and $\Delta$ is the 0–1 gate function defined in Eq. (3). Our optimization objective is maximizing the log-likelihood

$$\underset{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}}{\text{argmax}} \sum_{U \in \widetilde{\mathcal{T}}_t^k} (y_{t+1}(U))^\intercal \log p(y_{t+1}(U); \quad (5)$$
$$\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}).$$

The parameters $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, $\mathbf{b}^{(1)}$, and $\mathbf{b}^{(2)}$ are learned from Eq.(5) via stochastic gradient descent with early stopping. In what follows we show SPNN learns the same parameters irrespective of graph isomorphisms (see Supplemental Material for proof).

**Theorem 1.** *SPNN is isomorphic invariant. That is, given two graph sequences $G_1, G_2$ and $G'_1, G'_2$, where $G_n$ is isomorphic to $G'_n$, then the learned parameters $\hat{\mathbf{W}}^{(1)}, \hat{\mathbf{W}}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$ are exactly the same for the graph sequences $(G_1, G_2)$ and $(G'_1, G'_2)$ (assuming the same random seed).*

## Relationship with Convolutional Neural Networks

Images are lattices, trivial topologies, while general graphs are complex. Fundamentally, a CNN computes the output of various filters over local neighborhoods. In SPNN, the filter is the pattern, which maps the local neighborhood (within $d+\delta$ hops away from the target subgraph) into a single value. The distinct patterns act on overlapping regions of the neighborhood, but the amount of overlap is nontrivial for non-lattices. At CNNs, pooling at the upper layers often act as a rotation-invariance heuristic. SPNN upper layers are isomorphic-invariant by construction and SPNN performs pooling at the inputs. Moreover, similar to CNNs, SPNN can be augmented by multiple layers of fully connected units between the pattern layer and the predicted target.

## Related Work

In what follows we classify the existing literature based on the main obstacles in designing supervised learning methods for dynamic subgraph heterogeneous tasks: **(a)** The varying sizes of the different node neighborhoods, **(b)** accounting for distinct nodes and edge labels in the neighborhood; **(c)** isomorphic-invariance of graph representations (permutations of nodes in the adjacency matrix should not affect the representation); **(d)** the graph evolution; **(e)** learns from a single graph and includes the dependence structure of induced subgraphs that share edges and non-edges.

There are no existing approaches that can address all the above challenges. Existing approaches can be classified in the following categories:

(1) *Compute canonical representations of the whole graph (e.g., kernel or embeddings).* These methods require multiple examples of a whole graph (rather than induced subgraphs). Examples include GraphNN (Dai, Dai, and Song 2016), diffusion-convolution neural networks (Atwood and Towsley 2016), and graph kernels, such as Orsini et al.(Orsini, Frasconi, and De Raedt 2015) and Yanardag and Vishwanathan (Yanardag and Vishwanathan 2015a; 2015b), which compare small graphs based on the existence or count of small substructures such as shortest paths, graphlets, etc.. These whole-graph methods, however, are designed to classify small independent graphs, and fail to account for the sample dependencies between multiple induced subgraphs that share edges and non-edges. These whole-graph classification methods, collectively, address challenges **(a)**, **(b)**, **(c)**, and **(d)**, but fail to account for **(e)**.

(2) *Compute canonical representations of small induced subgraphs of the original graph*, (e.g., PATCHY (Niepert, Mohamed, and Konstantin 2016)) offers a convolutional neural network graph kernel that only addresses challenges **(b)** and **(c)**, but does not address **(d)** and **(e)**, and needs to pad features with zeros or arbitrarily cut neighbors from the feature vector, thus, not truly addressing **(a)**.

(3) *Compute isomorphism-invariant metrics over the graph*, such as most graph kernels methods, various diffusions (e.g., node2vec (Grover and Leskovec 2016), PCRW (Lao and Cohen 2010), PC (Sun et al. 2011a), deepwalk (Perozzi, Al-Rfou, and Skiena 2014), LINE (Tang et al. 2015), DSSM (Heck and Huang 2014), and deep convolutional networks for graph-structured data (Bruna et al. 2013; Henaff, Bruna, and LeCun 2015)), which address problems **(a)** and **(c)** but not **(b)**, **(d)** (specially because of edge labels), and **(e)**.

(4) *Perform a tensor factorization*, (e.g., RESCAL (Nickel, Tresp, and Kriegel 2011) and extensions (Nickel et al. 2015; Nickel, Rosasco, and Poggio 2016)), which addresses problems **(a)** and **(b)** but not **(c)**, **(d)**, and **(e)**. These methods are tailored specifically for the task of link prediction in heterogeneous graphs and are widely used.

To the best of our knowledge, SPNN is the only supervised learning method designed to predict subgraph evolution. Moreover, it addresses all the above challenges: **(a)** the SPNN representation uses subgraph patterns and their local densities, which are features that do not vary in size irrespective of the subgraph neighborhood; **(b)** SPNN not only accounts for labels but also accounts for how these labels are located within the network structure; **(c)** the learned SPNN model is invariant to isomorphisms (Thm. 1); and **(d)** SPNN is designed to learn the evolution of subgraph patterns; **(e)** the SPNN neural network structure assumes a Markov blanket that accounts for dependencies between subgraphs through their shared edges in a high-order network. While this procedure still does not guarantee we can break the graph into i.i.d. induced subgraphs, our empirical results show that it significantly outperforms models whose Markov blankets do not contain such high-order structures.

Other classical link prediction methods that can also be adapted to subgraph link prediction tasks. These methods (Dong et al. 2014; Lichtenwalter, Lussier, and Chawla 2010) use a wide variety of edge features, including pairwise features such as Adamic-Adar score, or path counts, such as from PCRW.

Separately, collective inference procedures (Richardson and Domingos 2006; Neville and Jensen 2007; Manfredotti 2009; Getoor and Mihalkova 2011), although traditionally evaluated at the node and edge level, can also include SPNN as a baseline predictor to be readily applied to dynamic subgraph tasks.

## Results

In this section we test the efficacy of SPNN, comparing it to other existing methods in the literature. We adapt these competing methods to the induced dynamic subgraph prediction task, as they are not designed for such tasks.

Our evaluation shows SPNN outperforms nine state-of-the-art methods in three real-world dynamic tasks. We show that the learned SPNN model weights can be used to draw insights into the predictions. We also evaluate SPNN across a variety of other synthetic dynamic tasks using static graphs (**Facebook** and **WordNet**), all reported in the appendix.

In the appendix, we also show that the architecture of SPNN also outperforms fully connected neural network layers for small training samples (both using the unique induced subgraph input features designed for SPNN, which explicitly model the subgraph dependencies). SPNN and fully connected layers with the same input vectors as SPNN have the same performance over larger training datasets.

### Empirical Results

**Datasets.** We use two representative heterogeneous graph datasets with temporal information. **DBLP** (Sun et al. 2011b) contains scientific papers in four related areas (AI, DB, DM, IR) with 14,376 papers, 14,475 authors, 8,920 topics, and 20 venues. We organize the dataset into authors, venues, and topics. Published papers represent links, for instance, two authors have a link at $G_n$ if they have co-authored a paper at time step $n$.
**Friendster** is a social network where user can post messages on each other's homepages. This dataset contains 14 millions of nodes and 75 million messages. Directed edges in this dynamic graph mark users writing on each other's message walls. The heterogeneous graph includes hometown, current locations, college, interests, and messages sent between users.

#### Subgraph Pattern Prediction Tasks.

a) **DBLP** task is to predict the evolution of 3-node subgraphs ($k = 3$): whether an author will publish in a venue and a topic that the author did not publish in the previous timestamp. This is a binary class problem, with pattern sets $\mathcal{Y}_1^3$and $\mathcal{Y}_2^3$ shown in Figure 1a. We use pattern sizes of $k + \delta = 4$. Our method generates 2700 features with 9 neurons in pattern layer as shown in Figure 4.

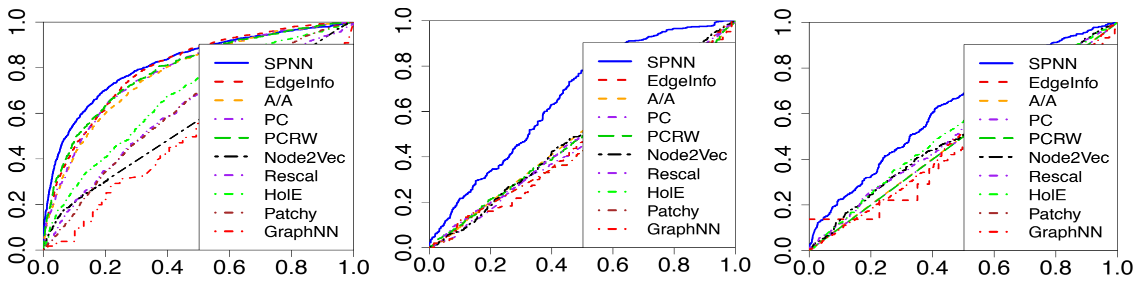


Figure 2: ROC curves (True Pos × False Pos): DBLP and Friendster tasks.

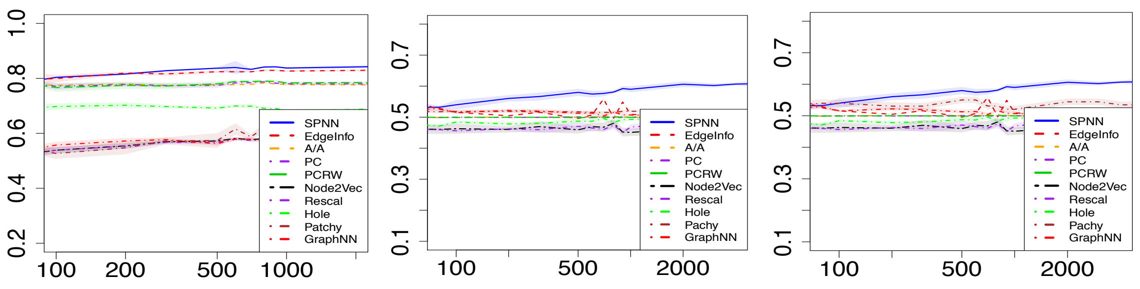


Figure 3: Learning curves (AUC×Training Size) w/shaded 95% confidence intervals.

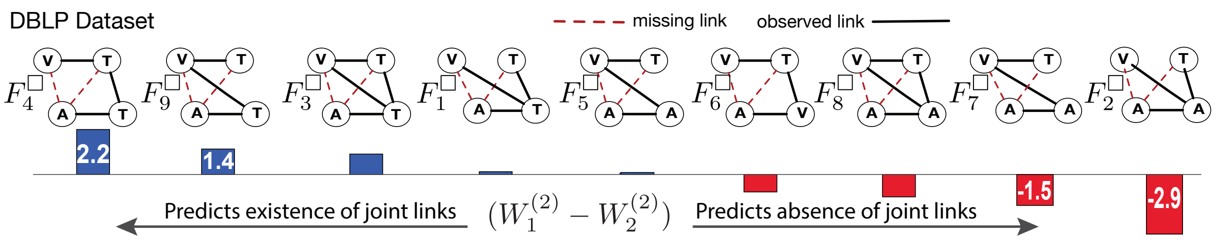

DBLP Dataset
missing link
observed link
2.2
1.4
-1.5
-2.9
Predicts existence of joint links
$(W_1^{(2)} - W_2^{(2)})$
Predicts absence of joint links

Figure 4: (DBLP task) Pattern layer $F_1^{\square}, \ldots, F_8^{\square}$, representing all connected subgraphs of $\mathcal{P}^{(4)}_{(\text{conn})}$ that appear in the training data. Bars show the difference between learned weights of Class 1 (whether both dashed links appear at time $t+1$) and Class 2 (everything else) for pattern $F_j^{\square}$. Pattern $F_4^{\square}$, when the author has published in a topic related to the venue, strongly predicts the appearance of both links. Pattern $F_2^{\square}$, when a co-author has published at the venue and topic of interest but not the author, strongly predicts the absence of the joint links.

| | **Independently Trained** **(Single Link Predictions)** | | | | | | **Jointly Trained Multi-Link Task** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EdgeInfo | PCRW | PC | N2V | Rescal | HolE | EdgeInfo | PCRW | PC | N2V | Rescal | HolE | Patchy | GraphNN | **SPNN** |
| DBLP | 0.811 | 0.786 | 0.783 | 0.567 | 0.611 | 0.681 | 0.830 | 0.782 | 0.788 | 0.582 | 0.611 | 0.690 | 0.627 | 0.571 | **0.846** |
| | ±0.012 | ±0.007 | ±0.012 | ±0.008 | ±0.025 | ±0.024 | ±0.007 | ±0.007 | ±0.014 | ±0.007 | ±0.025 | ±0.024 | ±0.003 | ±0.021 | ±0.011 |
| Friendster | 0.530 | 0.516 | 0.509 | 0.512 | 0.521 | 0.513 | 0.502 | 0.516 | 0.515 | 0.524 | 0.502 | 0.506 | 0.519 | 0.521 | **0.690** |
| (Activity) | ±0.088 | ±0.007 | ±0.006 | ±0.011 | ±0.031 | ±0.006 | ±0.007 | ±0.012 | ±0.012 | ±0.018 | ±0.012 | ±0.013 | ±0.010 | ±0.023 | ±0.008 |
| Friendster | 0.568 | 0.501 | 0.501 | 0.501 | 0.558 | 0.501 | 0.501 | 0.502 | 0.552 | 0.540 | 0.521 | 0.530 | 0.547 | 0.523 | **0.607** |
| (Structure) | ±0.011 | ±0.002 | ±0.002 | ±0.003 | ±0.009 | ±0.002 | ±0.004 | ±0.002 | ±0.019 | ±0.017 | ±0.017 | ± 0.021 | ± 0.025 | ±0.019 | ±0.017 |

Table 1: Max Area Under Curve (AUC) scores of SPNN against baselines.

b) **Friendster Activity** task predicts the increase in activity in weighted 4-node subgraphs: whether the total number of messages sent between four users, which are connected in the current time interval ($G_2$), increases in the next time interval ($G_3$). This is a binary classification task. The class set $\mathcal{Y}_1^3$ contains all subgraphs in the training data where the total number of messages between nodes increases between consecutive snapshots. The class set $\mathcal{Y}_2^3$ contains all other possible subgraphs that appear in the training data. Our method generates 1230 features with 30 neurons in the pattern layer.

c) **Friendster Structure** task predicts the evolution of 4-

node subgraphs: whether four friends who were weakly connected by three edges in the previous timestamp ($G_2$) will not send any messages in the next time stamp (i.e., be disconnected in $G_3$). This is a binary classification task. The class set $\mathcal{Y}_1^3$ contains the empty subgraph. The class set $\mathcal{Y}_2^3$ contains all subgraphs with at least one edge. Features are same as those in Friendster Activity.

To learn a predictive model of subgraph evolution, we divide the data into three temporal graphs $G_1, G_2, G_3$. The training set $\mathcal{T}_1^3$ comprises 3-node or 4-node subgraphs from $G_1$ with class labels $\mathbf{y}$ determined from $G_2$, and the test set $\mathcal{T}_2^3$ comprises subgraphs from $G_2$ with class labels from $G_3$. Since DBLP is a dynamic network with timestamps, we construct $G_1$ from the data in 2003–2004, $G_2$ from 2005–2006, and $G_3$ from 2007–2008. For Friendster, we construct $G_1$ from data in Jan 2007–April 2007. $G_2$ from May 2007–Aug 2007, and $G_3$ from Sep 2007–Dec 2007. We selected year 2007 because it is the most active time period for Friendster.

**Baselines.** We compare our approach to the nine methods discussed in Related Work. Five methods use isomorphic-invariant measures over the graph: (i) **AA**: Adamic–Adar score only (Adamic and Adar 2003); (ii) **EdgeInfo**: Uses all edge features listed in (Lichtenwalter, Lussier, and Chawla 2010); (iii) **PC**: Path counts (a.k.a. metapaths) (Sun et al. 2011a); (iv) **PCRW**: Path constrained random walk (Lao and Cohen 2010); (v) **Node2Vec**: Node embedding (Grover and Leskovec 2016). Two methods perform tensor factorizations: (vi) **Rescal**: Rescal embedding (Nickel, Tresp, and Kriegel 2011); (vii) **HolE**: Holographic embedding (Nickel, Rosasco, and Poggio 2016). One method computes canonical representations of small induced subgraphs of the original graph; (viii) **Patchy**: Patchy CNN graph kernel (Niepert, Mohamed, and Konstantin 2016); (ix) **GraphNN**: Embedding Mean-Field Inference (Dai, Dai, and Song 2016).

The above baselines, except **Patchy** and **GraphNN**, are originally intended to predict single missing links rather than make joint link predictions. We consider two different variants of the methods to apply the baselines to our joint link prediction tasks. The **Independent** approach trains separate classifiers, one for each link independently, and then combines the independent predictions into a joint prediction. The **Joint** approach concatenates the features of the multiple links into a single subgraph feature, then uses a classifier over the subgraph feature to make joint link predictions.

Moreover, these baselines, which are not developed for subgraph evolution tasks, generally achieve very poor predictive performance in a real temporal task that uses graphs $G_1$ and $G_2$ to predict $G_3$. Consider, for instance, the two distinct embeddings that **Node2Vec**, **Rescal**, and **HolE** assign to same nodes in $G_1$ and $G_2$ due to changes in the graph topology between $G_1$ and $G_2$. In order to use **Node2Vec**, **Rescal**, and **HolE** to predict links in dynamic graphs, we first learn node embeddings over $G_1$ and train a Multilayer Perceptron to predict links in $G_2$. Using this trained classifier, we again use the node embeddings of $G_1$ to predict the new links in $G_3$, and this improves their classification performance.

**Implementation.** We implement SPNN in Theano. The loss function is the negative log likelihood plus L1 and L2 regularization penalties over the parameters, both with regularization penalty 0.001. We train SPNN using stochastic gradient descent over a maximum of 30000 epochs and learning rate 0.01. 20% of the training examples are separated as validation for early stopping. All the data has the same amount of positive and negative examples. Source code is available at https://github.com/PurdueMINDS/SPNN.

**Comparison to Baselines.** Figure 2a-c shows the ROC curves of SPNN and baselines to predict balanced classes. We use 1000 induced subgraphs for training and 2000 induced subgraphs for testing (in all DBLP, Friendster Activity and Friendster Structure tasks). Since the testing sets have the same number of positive and negative examples, AUC scores are meaninful metrics to compare the models. SPNN outperforms all baselines in all tasks. Figure 3 shows the learning curves where training set sizes vary from 100 to 2000 subgraphs. Note that SPNN consistently achieves the best AUC scores. We summarize our results in Table 1, where we see that SPNN has significantly better AUC scores than the baselines over all tasks and datasets.

Table 1 also compares the performance of the **Independent** and **Joint** prediction approaches. Most methods show similar performance in both their **Independent** and **Joint** variants. This is likely due to the fact that the pair-wise similarity methods model link formation independently. Thus, the joint representation makes no difference in the two approaches. For low-rank decomposition methods (such as Rescal and HolE), we speculate that this is because edges are conditionally independent given the model, and, thus, they are unable to learn good low-dimensional embeddings for subgraph tasks where missing edges are dependent given the model.

| | DBLP | Friendster Activity | Friendster Structure |
|---|---|---|---|
| Rescal | 47.3min | 28h32min | 28h33min |
| HolE | 43.5min | 26h21min | 26h22min |
| node2vec | 2.9min | 3h51min | 3h51min |
| SPNN | 3.6min | 9min | 9min |

Table 2: Time to sample 1000 examples+training time.

Finally, Table 2 shows the wall-clock execution times of SPNN against the baselines HolE, Rescal, and Node2Vec. The server is an Intel E5 2.60GHz CPU with 512 GB of memory. Note that HolE and Rescal must compute an embedding for the entire graph, regardless of the number of subgraph examples. SPNN is orders of magnitude faster than HolE and Rescal and one order of magnitude faster than Node2Vec in the three tasks. Training SPNN takes around 90 seconds to sample and construct features for four-node subgraphs in DBLP, and 9 minutes for five-node subgraphs in Friendster. The significant difference in execution time is rooted in how long it takes to collect the induced subgraphs to train our model. For the relatively small two-year-sliced of DBLP, we enumerate all possible subgraphs and sample 1000 from them. For Friendster Activity and Friendster Structure tasks, we use the connected induced subgraph sampling method of Wang et al. (Wang et al. 2014) with an added bias to sample induced subgraphs of interest. In the worst case, learning SPNN takes $O(h|\mathcal{Y}||A|^k|R|^{k^2})$ time per iteration per training example, where $h = |\cup_n \mathcal{P}_n^{k+\delta}(\mathcal{T}_n^{(\text{sample})})|$ is the number of subgraph patterns in the pattern layer, $|\mathcal{Y}|$ is the number of distinct patterns in subgraph classes, $|A|$ is the number of node classes, and $|R|$ is the number of edge classes.

**Interpreting SPNN results.** Unlike most link prediction methods, SPNN's parameters are interpretable so that we can easily make sense of the predictions. Figure 4 shows the weight difference $W_1^{(2)}(j) - W_2^{(2)}(j)$ in SPNN's pattern layer between Class 1 and Class 2 for patterns $F_j^\square$ in the DBLP task. Large positive values indicate subgraph patterns that encourage the appearance of both dotted links while large negative values indicate patterns that discourage the appearance of both dotted links. Figure 4 caption details the examples of patterns $F_4^\square$ and $F_2^\square$.

## Conclusions

Our work is a first step in the development of more interpretable models, features, and classifiers that can encode the complex correlations between graph structure and labels. SPNN predicts induced subgraph evolution in heterogeneous graphs and generalizes a variety of existing tasks. Our results show SPNN to consistently achieve better performance than competing approaches. In future work we will develop a collective classification method to create a joint classifier (e.g., (Richardson and Domingos 2006; Neville and Jensen 2007)) for SPNN, as our approach can be used as a local conditional model for joint prediction.

## Acknowledgments

The authors would like to thank Carlos Teixeira for kindly giving us early access to his subgraph sampling software. This work was sponsored by the ARO, under the U.S. Army Research Laboratory, and the NSF under contract numbers W911NF-09-2-0053, IIS-1149789, IIS-1546488, and IIS-1618690.

## Supplementary Material

*Proof of Theorem 1.* In this proof we show that SPNN's input features, the training data, and the convolutional architecture all have a canonical representation invariant to isomorphisms. To this end, we show that: (a) the features of each training example $U \in \tilde{\mathcal{T}}_t^k$ have a canonical representation invariant to isomorphisms; (b) the training data $\tilde{\mathcal{T}}_t^k$ used in our stochastic gradient descent algorithm also has a canonical representation; and finally, (c) the neural network structure also has a canonical representation invariant to isomorphisms.

**(a)** The features of each training example $U \in \tilde{\mathcal{T}}_t^k$ are the vectors $\phi(U, F^\square, G_t)$ introduced in Eq.(4) for different patterns $F^\square \in \mathcal{P}_{(conn)}^{(k+\delta)}$ that appear in the training data. All we need to show is that vector $\phi$ has a canonical order invariant to graph isomorphisms. Observing Eq.(4), the $i$-th element of $\phi$, $(\phi(U, F^\square, G_t))_i$, has a canonical order as we can impose a canonical order on $\mathcal{P}_{(conn)}^{(k+\delta)}$ (e.g., lexicographic on the edges (Huan, Wang, and Prins 2003)). The value inside $(\phi(U, F^\square, G_t))_i$ is also clearly invariant to isomorphisms as it is the isomorphism density.

**(b)** The training data $\tilde{\mathcal{T}}_t^k$ are subgraphs of $G_t$ and, thus, also have a canonical representation via lexicographical ordering (Huan, Wang, and Prins 2003).

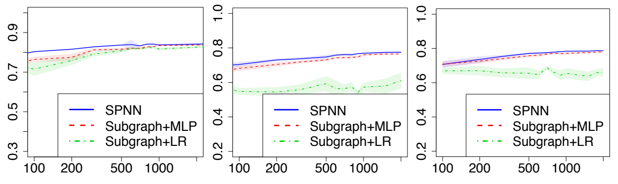
**(c)** As $\mathcal{P}_{(conn)}^{(k+\delta)}$ has a canonical order, so does the hidden layer of SPNN. Moreover, the $\Gamma$'s are similarly ordered.

The induced subgraphs of the training examples of the two isomorphic graphs $G_1$ and $G_1'$ have the same class labels, as the class labels are by definition isomorphic invariant. As there are canonical orderings of the data, features, class labels, and model structure that are invariant to isomorphic transformations of the graphs, and the stochastic gradient descent has the same random seed for all graphs, we conclude that SPNN must learn the same parameters. $\square$

**Link prediction on synthetic datasets** Besides the datasets with sequential information like DBLP and Friendster, we also test our proposed method on other famous heterogeneous datasets.

**Facebook** is a sample of the Facebook users from one university. The dataset contains 75,000 nodes and 8 million links. The heterogeneous graph includes friendship connections, user groups, political and religious views. **WordNet** is a knowledge graph that groups words into synonyms and provides lexical relationships between words. The WN18 dataset is a subset of WordNet, containing 40,943 entities, 18 relation types, and 151,442 triplets.

As discussed in Results Section, in order to learn a predictive model of subgraph evolution, we divide the data into three temporal graphs $G_1, G_2, G_3$. The Facebook and WordNet graphs are not dynamic, so we set $G_3$ to be the full network, and then randomly remove the links from $10\%$ of the subgraphs in Figure 7 (1)-(2) to construct $G_2$. Another $10\%$ are removed from $G_2$ to construct $G_1$.



(a) DBLP (b) Facebook (c) WordNet

Figure 5: Sequence Graph Learning curves (AUC$\times$Training Size) compared to logistic regression and MLP (w/shaded $95\%$ conf. intv.).

|  | Independently Trained (Single Link Predictions) |  |  |  |  |  | Jointly Trained Multi-Link Task |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | EdgeInfo | PCRW | PC | N2V | Rescal | HolE | EdgeInfo | PCRW | PC | N2V | Rescal | Hole | Patchy | SPNN |
| Facebook Dynamic | 0.748 (±0.014) | 0.738 (±0.009) | 0.725 (±0.011) | 0.526 (±0.011) | 0.523 (±0.017) | 0.750 (±0.007) | 0.747 (±0.003) | 0.723 (±0.009) | 0.725 (±0.012) | 0.527 (±0.011) | 0.632 (±0.007) | 0.746 (±0.006) | 0.510 (±0.006) | **0.774** (±0.015) |
| WordNet Dynamic | 0.606 (±0.028) | 0.553 (±0.003) | 0.551 (±0.004) | 0.528 (±0.018) | 0.586 (±0.009) | 0.618 (±0.009) | 0.639 (±0.023) | 0.551 (±0.003) | 0.553 (±0.003) | 0.524 (±0.018) | 0.586 (±0.009) | 0.611 (±0.009) | 0.574 (±0.038) | **0.786** (±0.006) |
| Facebook Static | 0.578 (±0.014) | 0.543 (±0.020) | 0.568 (±0.011) | 0.781 (±0.021) | 0.665 (±0.017) | 0.674 (±0.016) | 0.703 (±0.011) | 0.592 (±0.028) | 0.521 (±0.018) | 0.781 (±0.011) | 0.664 (±0.021) | 0.672 (±0.017) | 0.522 (±0.006) | **0.866** (±0.0014) |
| WordNet Static | 0.936 (±0.006) | 0.695 (±0.007) | 0.798 (±0.005) | 0.997 (±0.002) | 0.997 (±0.001) | 0.996 (±0.001) | 0.861 (±0.007) | 0.816 (±0.003) | 0.803 (±0.003) | 0.996 (±0.001) | 0.992 (±0.001) | 0.996 (±0.001) | 0.990 (±0.001) | **0.998** (±0.001) |

Table 3: Max Area Under Curve (AUC) scores of SPNN against baselines.

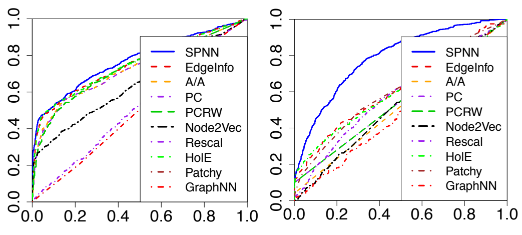
|  | DBLP | Facebook | WordNet |
|---|---|---|---|
| Rescal | 47.3min | 55h43.2min | 2h58.1min |
| HolE | 43.5min | 58h32.4min | 2h53.3min |
| node2vec | 2.9min | 74.0min | 10.0min |
| SPNN | 3.6min | 3.0min | 14.3min |

Table 4: Time to sample 1000 examples + learning time.

Figure 6a-b shows the ROC curves of SPNN and the baselines with 1000 training induced subgraphs and 2000 test induced subgraphs for Facebook and WordNet. SPNN outperforms all baselines in all tasks. Figure 8 shows the learning curves where training set sizes vary from 100 to 2000 subgraphs. Note that SPNN consistently achieves the best AUC scores. We summarize our results in first two rows of Table 3, where we see that SPNN has significantly better AUC scores than the baselines over all tasks and datasets.

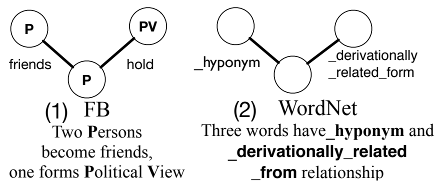
**Understanding Performance Gains.** To measure both the effect of (a) our induced isomorphism density features and (b) our sparse neural network architecture we compare SPNN against a logistic regression with the same input features as SPNN. The L2 regularized logistic regression verifies two things: (a) whether the deep architecture of SPNN is useful for our prediction task and (b) whether the induced isomorphism density features are more informative for our tasks than the Node2Vec, PCRW, Path Counts, and Edge features. The learning curves in Figure 5 show both (i) the benefit of one extra layer in the neural network and (ii) the gain in our features by contrasting the logistic regression against the learning curves of Figure 8.

The multi-layer perceptron (MLP) and SPNN differ in



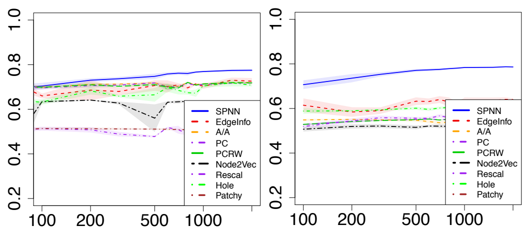
(a) Facebook (b) WordNet

Figure 6: ROC curves (True Pos × False Pos): Facebook and WordNet tasks in manually generated dynamic graphs



P
friends
P
PV
hold
(1) FB
Two Persons become friends, one forms Political View
_hyponym
_derivationally_related_form
(2) WordNet
Three words have _hyponym and _derivationally_related_from relationship

Figure 7: Facebook and WordNet Prediction tasks

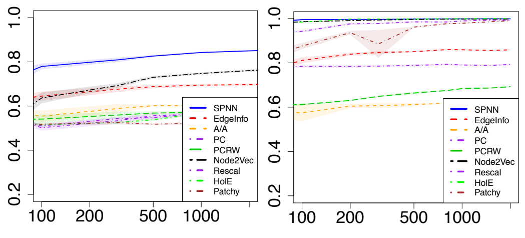


(a) Facebook (b) WordNet

Figure 8: Learning curves (AUC×Training Size) w/shaded 95% confidence intervals for dynamic Facebook and WordNet.

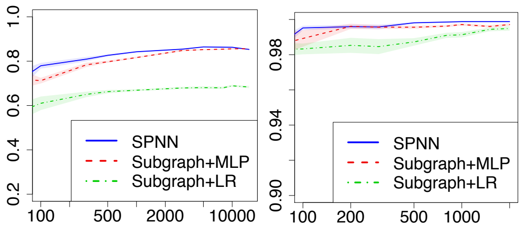
that MLP's input layer and hidden layer are fully connected. The MLP will help us test whether SPNN's sparse architecture is a good regularizer. The learning curves in Figure 5 show that SPNN outperforms MLP in majority cases with rare cases which have similar but not worse performance. This shows that the SPNN sparse architecture is indeed a good regularizer for the joint link prediction problem.

**Subgraph prediction in static graphs.** The experiments in Results Section has showed that our proposed method outperforms the state of the art in subgraph prediction on dynamic graphs. Our method can also predict missing links in static graphs such as Facebook and WordNet datasets without timestamps. 50% of the edges which belong to the two specified edge types in subgraph tasks shown in Figure 7 are removed randomly. To obtain positive examples, we sample or enumerate 4-node induced subgraphs $\mathcal{T}_1^{\square(4)}(3)$ which contains the removed subgraph. Randomly sample same amount of 4-node subgraphs which do not contain the removed structure as negative examples. Last two rows of Table 3 shows the performance against competing methods to predict subgraphs in static graphs. Figure 9 shows the learning curves. Both of these figures show that our pro-



(a) Facebook (b) WordNet

Figure 9: Learning curves (AUC×Training Size) of SPNN against competing methods in Static Graph (w/shaded 95% conf. intv.).



(a) Facebook (b) WordNet

Figure 10: Learning curves comparing SPNN to logistic regression and MLP in Static Graph.

posed SPNN consistently achieves the best performance, compared to all other methods. Figure 10 shows that our proposed SPNN sparse architecture is indeed a good regularizer for the joint link prediction problem in static graphs.

## References

Adamic, L., and Adar, E. 2003. Friends and neighbors on the web. *Social Networks* 25(3).

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *NIPS*.

Benson, A. R.; Gleich, D. F.; and Leskovec, J. 2016. Higher-order organization of complex networks. *Science* 353(6295).

Borgs, C.; Chayes, J. T.; Lovász, L.; Sós, V. T.; and Vesztergombi, K. 2008. Convergent Sequences of Dense Graphs I: Subgraph Frequencies, Metric Properties and Testing. *Advances in Mathematics*.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral Networks and Locally Connected Networks on Graphs. In *ICLR*.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2702–2711.

Dong, X.; Gabrilovich, E.; Heitz, G.; Horn, W.; Lao, N.; Murphy, K.; Strohmann, T.; Sun, S.; and Zhang, W. 2014. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. *KDD*.

Getoor, L., and Mihalkova, L. 2011. Learning statistical models from relational data. In *SIGMOD*, 1195. ACM Press.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*.

Heck, L., and Huang, H. 2014. Deep learning of knowledge graph embeddings for semantic parsing of twitter dialogs. In *GlobalSIP*. IEEE.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep Convolutional Networks on Graph-Structured Data. *arXiv:1506.05163v1*.

Huan, J.; Wang, W.; and Prins, J. 2003. Efficient mining of frequent subgraphs in the presence of isomorphism. In *ICDM*. IEEE Comput. Soc.

Lao, N., and Cohen, W. W. 2010. Relational retrieval using a combination of path-constrained random walks. *Mach. Learn.* 81(1).

Liben-Nowell, D., and Kleinberg, J. 2007. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.* 58(7).

Lichtenwalter, R. N.; Lussier, J. T.; and Chawla, N. V. 2010. New perspectives and methods in link prediction. In *KDD*.

Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; and Zhu, X. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. *AAAI*.

Lovász, L., and Szegedy, B. 2006. Limits of dense graph sequences. *J. Comb. Theory, Ser. B*.

Manfredotti, C. 2009. Modeling and inference with relational dynamic bayesian networks. In *Advances in artificial intelligence*. Springer.

Neville, J., and Jensen, D. 2007. Relational dependency networks. *JMLR*.

Nickel, M.; Murphy, K.; Tresp, V.; and Gabrilovich, E. 2015. A Review of Relational Machine Learning for Knowledge Graphs. *IEEE*.

Nickel, M.; Rosasco, L.; and Poggio, T. 2016. Holographic embeddings of knowledge graphs. In *AAAI*. AAAI Press.

Nickel, M.; Tresp, V.; and Kriegel, H.-P. 2011. A Three-Way Model for Collective Learning on Multi-Relational Data. *ICML*.

Niepert, M.; Mohamed, N.; and Konstantin, N. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*.

Orsini, F.; Frasconi, P.; and De Raedt, L. 2015. Graph Invariant Kernels. In *IJCAI*.

Paranjape, A.; Benson, A. R.; and Leskovec, J. 2017. Motifs in Temporal Networks. In *WDSM*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM.

Rahman, M., and Al Hasan, M. 2016. Link prediction in dynamic networks using graphlet. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 394–409. Springer.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Mach. Learn.*

Sun, Y.; Barber, R.; Gupt, M.; Aggarwal, C.; and Han, J. 2011a. Co-author relationship prediction in heterogeneous bibliographic networks. In *ASONAM*.

Sun, Y.; Han, J.; Yan, X.; Yu, P. S.; and Wu, T. 2011b. PathSim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB* 4(11).

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*. ACM.

Wang, P.; Lui, J. C. S.; Ribeiro, B.; Towsley, D.; Zhao, J.; and Guan, X. 2014. Efficiently Estimating Motif Statistics of Large Networks. *ACM TKDD* 9(2).

Xu, J.; Wickramarathne, T. L.; and Chawla, N. V. 2016. Representing higher-order dependencies in networks. *Science Advances* 2(5).

Yanardag, P., and Vishwanathan, S. 2015a. A Structural Smoothing Framework For Robust Graph Comparison. In *NIPS*.

Yanardag, P., and Vishwanathan, S. 2015b. Deep Graph Kernels. In *KDD*.