Neural Embedding Propagation on Heterogeneous Networks

Carl Yang*, Jieyu Zhang*, Jiawei Han
University of Illinois, Urbana Champaign, 201 N Goodwin Ave, Urbana, Illinois 61801, USA
{jiyang3, jieyuz2, hanj}@illinois.edu

***Abstract*—Classification is one of the most important problems in machine learning. To address label scarcity, semi-supervised learning (SSL) has been intensively studied over the past two decades, which mainly leverages data affinity modeled by networks. Label propagation (LP), however, as the most popular SSL technique, mostly only works on homogeneous networks with single-typed simple interactions. In this work, we focus on the more general and powerful heterogeneous networks, which accommodate multi-typed objects and links, and thus endure multi-typed complex interactions. Specifically, we propose *neural embedding propagation* (NEP), which leverages distributed embeddings to represent objects and dynamically composed modular networks to model their complex interactions. While generalizing LP as a simple instance, NEP is far more powerful in its natural awareness of different types of objects and links, and the ability to automatically capture their important interaction patterns. Further, we develop a series of efficient training strategies for NEP, leading to its easy deployment on real-world heterogeneous networks with millions of objects. With extensive experiments on three datasets, we comprehensively demonstrate the effectiveness, efficiency, and robustness of NEP compared with state-of-the-art network embedding and SSL algorithms.**

I. Introduction

The last decade has witnessed tremendous success of deep learning models, most of which highly rely on the availability of large amounts of training data [1], [2]. *Semi-supervised learning* (SSL) [3], [4], which is essentially close to the recent popular scheme of *few-shot learning* [5], [6], [7], naturally aims at alleviating such reliance on training data. Arguably the most classic model for SSL is based on transductive inference on graphs, *i.e.*, *label propagation* (LP) [3], [4]. Given a mixed set of labeled and unlabeled data points (*e.g.*, images), LP firstly constructs a homogeneous affinity network (*e.g.*, a k-nearest-neighbor adjacency matrix), and then propagates labels on the network. Due to its simplicity and effectiveness, LP has found numerous industrial applications [8], [9] and attracted various following-up research [10], [11], [12].

While SSL is well studied on homogeneous networks, in the real world, however, data are often multi-typed and multi-relational, which can be better modeled by heterogeneous networks [13], [14]. For example, in a movie recommendation dataset [15], the basic units can be users, movies, actors, genres and so on, whereas in a place recommendation dataset [16], [17] objects can be users, places, categories, locations, *etc*. Moreover, knowledge bases such as Freebase and YAGO can also be naturally modeled by heterogeneous networks, due to their inherent rich types of objects and links. As a powerful model, heterogeneous network enables various tasks like the classification and recommendation of movies and places, as well as relational inference in knowledge bases, where SSL is highly desired due to the lack of labeled data. However, trivial adoptions of LP on heterogeneous networks by suppressing the type information are not ideal, since they do not differentiate the functionalities of multiple types of objects and links.

To leverage the multi-relational nature of heterogeneous networks, the concept of *meta-paths* (or *meta-graphs*, as a more general notion) has been proposed and widely used by existing models on heterogeneous networks [18], [19], [20]. For the particular problem of SSL, [14], [21], [22], [13] have also leveraged meta-paths to capture the different semantics among targeted types of objects. However, as pointed out in [23], [16], the assumption that all useful meta-paths can be pre-defined by humans is often not valid, and exhaustive enumeration and selection over the exponential number of all possible ones is impractical. Therefore, existing methods considering a fixed set of meta-paths cannot effectively capture and differentiate various object interactions on heterogeneous networks.

*Both authors contributed equally to this work.

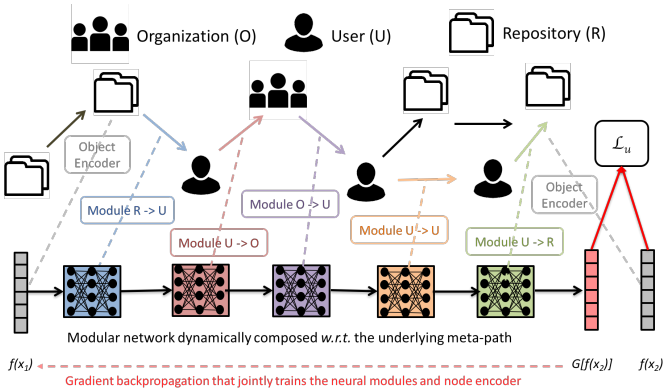

Fig. 1. A running toy example of NEP on the GitHub heterogeneous network.

In this work, to address the limitations of the existing works, we propose a novel NEP (*Neural Embedding Propagation*) framework for SSL over heterogeneous networks. Figure 1 gives a running toy example of NEP, which is a *powerful* yet *efficient* neural framework that coherently combines an *object encoder* [16], [17] and a *modular network* [24], [25]. It leverages the compositional nature of meta-paths and trivially generalizes to attributed networks.

In Figure 1, the object encoder can be simply implemented as an embedding look-up table. Trained together with a parametric predictive model (*e.g.*, a multi-layer perceptron (MLP)), it computes a mapping between object representations in a latent embedding space and object labels given as supervision. Such embeddings capture the correlations among object labels, and alleviates their inherent sparsity and noise. We find it also important to allow the embedding of labeled objects to change along training, which indicates that unlabeled data may even help improve the modeling of labeled data.

One step further, to capture the complex interactions on different types of links, we cast each of them as a unique differentiable neural network module (*e.g.*, also an MLP). Different meta-paths then correspond to unique modular networks, which are dynamically composed through stacking the corresponding neural network layers *w.r.t.* the particular link types along the paths. During the training of NEP, each time starting from a particular object, to mimic the process of LP, we *propagate* its *label* along a particular sampled path, by feeding its *object embedding* into the corresponding *modular network*. An ℓ_2-loss is computed between the *propagated* embedding and the original embedding on the end object, to require proper

smoothness between the connected objects. Then the gradients are back propagated along the path to update both the corresponding neural modules and the object encoder.

Due to the expressiveness of neural networks, NEP is able to automatically discover the functionalities of different types of links and dynamically model their common compositions (*i.e.*, meta-paths) on-the-fly based on uniform random walks, which allows us to abandon the explicit consideration of a limited set of meta-paths but rather model them in a data-driven way. Finally, as non-linearity can be easily added into the MLP-based neural network modules, NEP can be more flexible with complex object interactions.

To further improve the efficiency of NEP, we design a series of intuitive and effective training strategies. Firstly, in most scenarios, we only care about the labels of certain targeted types of objects. This allows us to only compute their embeddings and sample the random paths only among them. Secondly, to fully leverage training labels, we reversely sample the random paths from labeled objects, which makes sure the propagation paths all end on labeled objects, so the propagated embeddings can directly encode high-quality label information. Finally, to boost training efficiency, we design a two-step path sampling approach, which essentially groups instances of the same meta-paths into mini-batches, so that the same modular network is instantiated and trained in each mini-batch, leading to 300+ times gain on efficiency as well as slight gain on effectiveness.

Our experiments are done on three real-world heterogeneous networks with millions of objects and links, where we comprehensively study the effectiveness, efficiency and robustness of NEP. NEP is able to achieve $23.2\% - 33.6\%$ relative gain on classification accuracy compared with the average scores of all baselines across three datasets, which indicates the importance of the proper modeling of complex object interactions on heterogeneous networks. Besides, NEP is also shown to be the most efficient regarding the leverage of training data and computational resources, while being robust towards hyper-parameters in large ranges. All code will be released upon the acceptance of this work.

II. Related Work and Preliminaries

A. Heterogeneous Network Modeling

Networks are widely adopted as a natural and generic model for interactive objects. Most of recent network models focus on the higher-order object interactions, since few interactions are independent of others. Arguably, the most popular ones include personalized page rank [26] and DeepWalk [27] based on random walks, LINE [28] and graph convolutional networks [12] leveraging the direct node neighborhoods, as well as higher-order graph cut [29] and graph kernels methods [30] considering small network motifs with exact shapes. All of them have stimulated various follow-up works, the discussion of which is beyond the scope of this work.

In the real world, objects have multiple types and interact in different ways, which leads to the invention of heterogeneous networks [31]. Due to its capacity of retaining rich representations of objects and links, it has drawn increasing research attention in the past decade and facilitated various downstream applications including link prediction [32], classification [33], clustering [34], recommender systems [35], outlier detection [36] and so on.

Since objects and links in heterogeneous networks have multiple types, the interaction patterns are much more complex. To capture such complex interactions, the tool of meta-path has been proposed and leveraged by most existing models on heterogeneous networks [18]. Traditional object proximity models measure the total strength of various interactions by counting the number of instances of different meta-paths between objects and adding up the counts with pre-defined or learned weights [18], [23], [19], [37], [22], [21], [38], [14], [39], whereas the more recent network representation learning methods leverage meta-path guided random walks to jointly model multiple interactions in a latent embedding space [20], [40], [15], [41], [13], [42]. However, the consideration of a fixed set of meta-paths, while helping regulate the complex interactions, largely relies on the quality of the meta-paths under consideration, and limits the flexibility of the model, which is unable to handle any interactions not directly captured by the meta-paths.

B. Semi-Supervised Learning

Semi-supervised learning (SSL) aims at leveraging both labeled and unlabeled data to boost the performance of various machine learning tasks. Among many SSL methods, the most classic and influential one might be label propagation (LP) [3], [4]. Its original version assumes the input of a small amount of labeled data and a data affinity network, either computed based on the distances among attributed objects, or derived from external data.To predict the labels of unlabeled data, it propagates the labels from labeled data based on the topology of the affinity network, with the smoothness assumption that nearby objects on the network tend to have similar labels. Due to the simplicity and effectiveness of LP, many follow-up works have been proposed to improve it, especially on the homogeneous network setting [10], [11], [12], [43].

SSL has also been studied in the heterogeneous network setting. The uniqueness of heterogeneous network is its accommodation of multi-typed objects and relations, thus leading to complex object interactions and propagation functions. Therefore, all SSL models on heterogeneous networks leverage a given set of meta-paths to regulate and capture the complex object interactions. For example, [21], [22] both use a set of meta-paths to derive multiple homogeneous networks and optimize the label propagation process on all of them, whereas [37], [14], [44], [45], [46] jointly optimize the weight of different meta-paths. [13], [42] simultaneously preserves the object proximities *w.r.t.* multiple meta-paths to learn a unique network embedding. However, besides the limitation of given set of meta-paths, they still only consider simple interaction patterns with linear propagation functions.

III. Neural Embedding Propagation

In this section, we describe our NEP (*Neural Embedding Propagation*) algorithm, which coherently combines embedding learning and modular networks into a powerful yet efficient SSL framework over heterogeneous networks.

A. Motivations and Overview

In this work, we study SSL over heterogeneous networks. Therefore, the input of NEP is a heterogeneous network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$ are the multi-typed objects and links, respectively. In general, $\mathcal{V}$ can be associated with $\{\mathcal{Y}, \mathcal{A}\}$, where object labels $\mathcal{Y}$ is often only available in a small subset $\mathcal{V}_l \subset \mathcal{V}$, and object attributes $\mathcal{A}$ can be available for all objects, part of all objects, or none of the objects at all. In this work, we focus on predicting the labels of all objects in $\mathcal{V}$ based on both $\mathcal{Y}$ and $\mathcal{E}$.

Before formally introducing the heterogeneous network setting, let us first consider SSL over homogeneous networks. Particularly, we aim to explain why LP is sufficiently effective in that situation.

In homogeneous networks, since all objects share a single type, it is legitimate for LP to directly put any labels to any objects in the network. Also, labels on a single type of objects are often

mutually exclusive and thus can be considered disjointly without a predictive model. Moreover, since all links share a single type, the only thing that can differ across links is their weight, which can be easily modeled by simple linear propagation functions.

To understand the unique challenges of SSL in the heterogeneous network setting, we firstly briefly review the definition of heterogenous networks as follows.

Definition III.1. *A heterogeneous network [31], [18] is a network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with multiple types of objects and links. Within $\mathcal{G}$, $\mathcal{V}$ is the set of objects, where each object $v \in \mathcal{V}$ is associated with an object type $\phi(v)$, and $\mathcal{E}$ is the set of links, where each link $e \in \mathcal{E}$ is associated with a link type $\psi(e)$. It is worth noting that a link type automatically defines the object types on its two ends.*

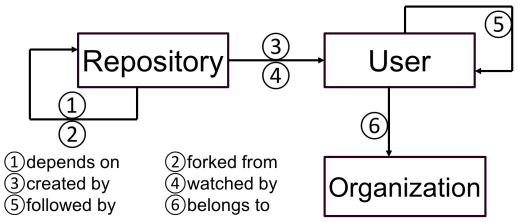

Fig. 2. Schema of the GitHub heterogeneous network.

Our toy example of GitHub data can be seen as a heterogeneous network, where the basic object types include user, repository and organization. The particular network schema is shown in Figure 2.

According to Definition III.1, in the heterogeneous network setting, due to the existence of multiple object types, labels of different types of objects can not be directly propagated, but they rather interact implicitly. For example, in our GitHub network in Figure 1, directly assigning the user label like "ios developer" to a repository object does not make much sense, but such a user label does indicate that the linked repositories might be more likely to be associated with labels like "written in objective c".

To capture such latent semantics and interactions of labels, as well as addressing their inherent noise and sparsity, we propose and design an object encoder to map various labels into a common embedding space (Section III.B). As a consequence, we propagate object embeddings instead of labels on the network, and a parametric predictive model is applied to map the embeddings back to labels upon prediction. Moreover, as we will show in more details later, this object encoder can be easily extended to incorporate the rich information in various object attributes.

In heterogeneous networks, different types of objects can interact in various ways, which obviously cannot be sufficiently modeled by simple weighted links. Consider our GitHub network in Figure 1, where users can "belong to" organizations and "create" repositories. The links derived by the "belong to" and "create" relations should thus determine different label propagation functions. For example, the labels of organizations might be something like "stanford university" or "google inc.", whereas those of repositories might be "written in objective c" or "tensorflow application". In this case, although the labels of both organizations and repositories can influence users' labels regarding "skills" and "interests", the mapping of such influences should be quite different. Moreover, consider the links even between the same types of objects, say, users and repositories. Since users can "create" or "watch" repositories, the different types of links should have different functions regarding label propagation. For example, when a user "creates" a repository, her labels regarding "skills" like "fluent in python" might strongly indicate the labels of the "created" repository like "written in python", but when she "watches" a repository, her labels regarding "interests" like "deep learning fan" will likely indicate the labels of the "watched" repository like "tensorflow application".

To model the multi-typed relations among objects, we propose to cast each type of links as a unique neural network module (Section III.C). The same module is reused over all links of the same type, so the number of parameters to be learned is independent of the size of the network, making the model efficient in memory usage and easy to train. These link-wise neural networks are jointly trained with the object encoders, so that the complex semantics in object labels (and possibly object attributes) can be well modeled to align with the various object interactions and propagation functions determined by different types of links.

One step further, as pointed out by various existing works, we notice that the higher-order semantics in heterogeneous networks can be regulated by the tool of meta-path, defined as follows.

Definition III.2. *A meta-path [31], [18] is a path defined on the network schema denoted in the form of $o_1 \xrightarrow{l_1} o_2 \xrightarrow{l_2} \cdots \xrightarrow{l_m} o_{m+1}$, where o and l are object types and link types, respectively. It represents a compositional relation between two given object types.*

Each meta-path thus captures a particular aspect of semantics. Continue with our example on the GitHub network in Figure 1. The meta-path of $\text{user} \xrightarrow{\text{creates}} \text{repository} \xrightarrow{\text{watched by}} \text{user}$ carries quite different semantics from $\text{user} \xrightarrow{\text{belongs to}} \text{organization} \xrightarrow{\text{includes}} \text{user}$. Thus, the two pairs of users at the ends of these two paths are similar in different ways, which are *composed* by the modular links along the paths and should imply different label propagation functions.

To fully incorporate the higher-order complex semantics in heterogeneous networks, we leverage the compositional nature of paths and propose to jointly train our link-wise neural network modules through randomly sampling paths on heterogeneous networks and dynamically constructing the neural modular networks corresponding to their underlying meta-paths (Section III.D). In this way, we do not require the input of a given set of useful meta-paths, nor do we need to enumerate all legitimate ones up to a certain size. Instead, we let the random walker compose arbitrary meta-paths during training, and automatically estimate their importance and functionalities regarding LP on-the-fly.

Finally, although NEP is powerful yet light in parameters, we deliberately designed a series of training techniques to further improve its efficiency (Section III.E). We also systematically and theoretically analyze the connections between NEP and various popular SSL algorithms, and briefly talk about several straightforward extensions of NEP left as future works (Section III.F).

B. Object Encoder of Labels and Beyond

Standard LP directly propagates labels on the whole network by assigning each object a label probability distribution. In this way, besides the label-object imcompatibility as we discussed before, they also ignore the complex label semantics and correlations. Moreover, labels in real-world datasets are often sparse and noisy, due to the high expense of high-quality label generation, which leads to the built-up of error rates during propagation.

To overcome these problems, instead of propagating labels, we propose to firstly encode various object labels into a common latent space, and then propagate the object embeddings on the network. To this end, we leverage the power of neural representation learning by

jointly training an object embedding function and a label prediction function for object encoding.

Particularly, we have the embedding $\mathbf{x}_i$ of object v_i as $\mathbf{x}_i = \mathbf{f}(v_i)$. In the simplest case, $\mathbf{f}(\cdot)$ can be implemented as a randomly initialized learnable embedding look-up table, *i.e.*, $\mathbf{x}_i = \mathbf{E}^T\mathbf{u}_i$, where $\mathbf{E} \in \mathbb{R}^{N\times K}$ is the embeddings of the total N objects on the network $\mathcal{G}$ into a K-dimensional latent space, and $\mathbf{u}_i$ is the one-hot vector representing the identity of v_i.

To encode various labels into a common latent space, we apply an MLP on the object embedding as a parametric label prediction model and impose a *supervised loss* in terms of cross-entropy on softmax classification *w.r.t.* ground-truth labels on labeled objects.

$$\mathcal{J}_l = -\sum_{i=1}^{M} \log p(v_i, y_i) = -\sum_{i=1}^{M} \log \frac{\exp(\mathbf{W}_l^{y_i}\tilde{\mathbf{x}}_i)}{\sum_{y\in\mathcal{Y}}\exp(\mathbf{W}_l^{y}\tilde{\mathbf{x}}_i)}, \quad (1)$$

where $M = |\mathcal{V}_l|$ is the number of labeled objects, y_i is the ground-truth label of object v_i, and $\mathcal{Y}$ is the set of all distinct labels on the network. It is trivial to encode multiple labels for a single object, by computing $\sum_{y\in\mathbf{y}_i} p(v_i, y)$.

Moreover, we have

$$\tilde{\mathbf{x}} = \mathbf{h}_n^{Q_n}(\ldots\mathbf{h}_n^1(\mathbf{x})\ldots), \quad (2)$$

where

$$\mathbf{h}_n^q(\mathbf{x}) = \text{ReLU}(\mathbf{W}_n^q\mathbf{h}_n^{q-1}(\mathbf{x}) + \mathbf{b}_n^q). \quad (3)$$

Q_n is the number of layers in the MLP, $\mathbf{W}_n^q$ and $\mathbf{b}^q$ are the parameters of the q-th layer, and $\mathbf{h}^0(\mathbf{x}) = \mathbf{f}(\mathbf{x})$. We use $\Theta_n = \{\mathbf{W}_l, \mathbf{W}_n, \mathbf{b}_n\}$ to denote all parameters in the MLP-based parametric prediction model. As we motivated above, such an MLP is useful in capturing the complex label semantics and correlations, and at the same time address the label noise and sparsity.

Note that, discussed above is a basic object encoder that only considers object labels. As to be shown in Section III.F, it is straightforward to extend this object encoder to consider the rich information of available attributes $\mathcal{A}$ associated with objects.

C. Type-Aware Link-Wise Modules

Now we consider the process of embedding propagation on heterogeneous networks, where multiple types of objects interact in rather complex ways. Our key insight here is, if we regard each link in the network as an influence propagation channel which allows the connected objects to influence each other, then different link types should naturally determine different propagation functions. To explicitly leverage this insight, we use a unique neural network to model the propagation functions of each type of links, which acts as a reusable module on the whole network.

Particularly, for each module, we still resort to the MLP of feedforward neural networks, due to its compatibility with the object encoder and representation power to model the complex label-link interactions, as well as model simplicity. For each link type $t \in \mathcal{T}$, we have

$$\mathbf{g}_t(\mathbf{x}) = \mathbf{h}_t^{Q_t}(\ldots\mathbf{h}_t^1(\mathbf{x})\ldots), \quad (4)$$

where

$$\mathbf{h}_t^q(\mathbf{x}) = \text{ReLU}(\mathbf{W}_t^q\mathbf{h}_t^{q-1}(\mathbf{x}) + \mathbf{b}_t^q). \quad (5)$$

$\forall t \in \mathcal{T}, Q_t$ is the number of layers in the MLP, $\mathbf{W}_t^q$ and $\mathbf{b}_t^q$ are the parameters of the q-th layer, and $\mathbf{h}_t^0(\mathbf{x}) = \mathbf{f}(\mathbf{x})$. We use $\Theta_m = \{\mathbf{W}_t, \mathbf{b}_t\}_{t\in\mathcal{T}}$ to denote all parameters in all of the MLP-based link-wise neural network modules. Note that, we use $\mathcal{T}$ to denote the set of all link types, and each link type is counted twice by considering the propagation directions. As we will show in the experiments, compared with linear MLP, non-linear MLP allows the model of object interactions to be more flexible and effective.

Equipped with such link-wise propagation functions, to mimic the process of LP from object v_i to object v_j through link e_{ij}, we simply input the object embedding $\mathbf{x}_i$ into the neural network module corresponding to link type $t = \psi(e_{ij})$, and get $\mathbf{g}_t(\mathbf{x}_i)$ as the propagated embedding. An *unsupervised loss* (*e.g.*, an ℓ_2-loss) is then computed between the propagated embedding of v_i on v_j and the current embedding of v_j to require the *label smoothness* among v_i and v_j, *conditioned* on their particular link type $\psi(e_{ij})$. Specifically, for all linked pairs of objects on the network, we have

$$\mathcal{J}_u = \sum_{e_{ij}\in\mathcal{E}} ||\mathbf{g}_{\psi(e_{ij})}(\mathbf{x}_i) - \mathbf{x}_j||_2^2. \quad (6)$$

Multiple links among the same pair of objects can also be trivially considered with our model by adding up all corresponding losses.

By combining the supervised loss in Eq. 1 and unsupervised loss in Eq. 6, we can simply arrive at the overall loss function of NEP, which implements SSL over a heterogeneous network as follows.

$$\mathcal{J} = \mathcal{J}_l + \lambda\mathcal{J}_u, \quad (7)$$

which shares the identical form with the general objective function of LP [3], [4] and various other SSL algorithms. By properly optimizing $\mathcal{J}$, we can jointly train our neural object encoder and link-wise modules, so that the embedding propagation along each link is jointly decided by both the end objects (particularly the propagated labels in our current model) and the link type.

D. Comprehensive Semantics with Path Sampling

We notice that most existing models on heterogeneous networks including the recent works on SSL [14], [13] all leverage the tool of meta-paths to capture fine-grained semantics regarding the higher-order interactions involving multiple object and link types. However, all of them explicitly model a limited set of meta-paths, which only leverages part of all complex interactions.

In this work, we leverage the *compositional nature* of paths, and propose to dynamically sample uniform random walks on heterogeneous networks and compose the corresponding modular neural networks during model training with ultimate flexibility on-the-fly. In this sense, our consideration of meta-paths is truly data-driven, *i.e.*, the sampled paths, while naturally preferring the more common and important underlying meta-paths in particular heterogenous networks, can actually also reveal any possible meta-paths. Therefore, we are able to avoid the explicit consideration of any limited sets of meta-paths and capture the comprehensive higher-order semantics in arbitrary heterogeneous networks.

When training NEP, instead of limiting the embedding propagation along direct links, we consider it along paths consisting of multiple links. Particularly, we revise the unsupervised loss function in Eq. 6 into

$$\mathcal{J}_u' = \sum_{p_{ij}\in\mathcal{P}} ||\mathbf{G}_{p_{ij}}(\mathbf{x}_i) - \mathbf{x}_j||_2^2, \quad (8)$$

where p_{ij} is a path sampled with uniform random walks on the heterogeneous network, and $\mathcal{P}$ is the set of all randomly sampled paths. Correspondingly, we have $\mathcal{J}' = \mathcal{J}_l + \lambda\mathcal{J}_u'$.

Definition III.3. *A uniform random walk in heterogeneous networks is a random walk that ignores object types. Particularly, on object v_i, the random walker picks the next object v_j to go to based on the uniform link distribution with $p(e_{ij}|v_i) = 1/deg(v_i)$, where $deg(\cdot)$ is the total number of all types of links that connect to v_i. We do not consider self-loops or restarts.*

Next we talk about the construction of $\mathbf{G}_{p_{ij}}$, by starting with the definition of a path p_{ij}.

Definition III.4. *A path p_{ij} is an ordered list $(v_i, e_1, e_2, \ldots, e_n, v_j)$, where v_i and v_j are the source and destination objects, respectively. $(e_1 \ldots, e_n)$ are the links along the path. n is the number of links in the path, and a path with n links is called a length-n path.*

With the link-wise neural network modules defined in Section III.C, we further leverage the idea of modular neural networks from visual question answering [24], [25], by dynamically constructing $\mathbf{G}_{p_{ij}}$ *w.r.t.* the underlying meta-path of p_{ij} as follows.

$$\mathbf{G}_{p_{ij}} = \mathbf{g}_{\psi(e_1)} \circ \mathbf{g}_{\psi(e_2)} \circ \cdots \circ \mathbf{g}_{\psi(e_n)}. \tag{9}$$

As we can see, by stacking the corresponding link-wise modules in the correct order, each meta-path now corresponds to a unique neural network model, where the components can be jointly trained and reused. As a consequence, each meta-path determines a unique learnable embedding propagation function, which further depends on the propagation functions of all of its component links. On one hand, the dynamically composed path-wise models capture the complex fine-grained higher-order semantics in the heterogeneous networks, while on the other hand, the learning of the link-wise modules is enhanced across the training based on various paths. As a result, NEP can be efficiently trained to deeply capture the comprehensive semantics and importance of any arbitrary meta-paths regarding the embedding propagation functions, which totally breaks free the requirements of given set of meta-paths and explicit search or learning for linear importance weights [40], [14], [22].

E. Further Efficiency Improvements

To further improve the efficiency of NEP, we design a series of intuitive and effective strategies.

Focusing on Targeted Types of Objects. Our NEP framework is designed to model heterogeneous networks with multiple types of objects, which naturally can be associated with multiple sets of labels. However, in some real-world scenarios, we only care about the labels of some particular *targeted* types of objects. For example, when we aim to classify repositories on GitHub, we are not explicitly interested in user and organization labels.

Due to this observation, we can aggressively simplify NEP by only computing the embeddings of targeted types of objects and subsequently constraining the random paths to be sampled only among them. We call this model NEP-target. Compared with NEP-basic, NEP-target allows us to significantly reduce the size of the embedding look-up tables in the object encoder by $35\% - 65\%$, which costs the most memory consumption, compared with other model parameters that are irrelevant to the network sizes. Moreover, since we focus on the embedding propagation among targeted types of objects, NEP-target can effectively save the time of learning the encodings of non-targeted types of objects, which leads to about 60% *shorter runtimes* until convergence compared with NEP-basic. Finally, it also helps to alleviate the built-up of noises and errors when propagating through multiple poorly encoded intermediate objects, which results in $12\% - 21\%$ *relative performance gain* compared with NEP-basic.

Note that, ignoring the embedding of non-targeted objects does not actually contradict with our model motivation, which is to capture the complex interactions among different types of objects. This simplification only works in particular scenarios like the ones we consider in this work, where we only care about and have access to the labels of particular types of objects, and the identities of non-targeted objects are less useful without the consideration of their labels and attributes. In this case, the only information that matters for the non-targeted types of objects is their types, which is sufficiently captured by our type-aware link-wise modules.

Fully Leveraging Labeled Data. By focusing on targeted types of objects, we have saved a lot of training time for learning the embeddings of non-targeted types of objects. However, on real-world large-scale networks, learning the embeddings of unlabeled targeted types of objects can still be rather inefficient. This is because the embeddings of most objects (*i.e.*, unlabeled objects) are meaningless at the beginning, and therefore the modeling of their interactions is also wasteful.

Our first insight here is, to fully leverage labeled data, we should focus on paths that include at least one labeled object, whose embedding directly encodes label information. Since our modular neural networks are reused everywhere in the network, the propagation functions of different links and paths captured around labeled objects are automatically applied to those among unlabeled objects. Moreover, due to the small diameter property of real-world networks [47], we assume that moderately long (*e.g.*, length-4) paths with at least one labeled object can reach most unlabeled objects for proper learning of their embeddings.

One step further, we find it useful to only focus on paths ending on labeled objects. The insight here is, according to Eq. 8, the ℓ_2-loss is computed between the propagated embedding $\mathbf{G}_{p_{ij}}(\mathbf{x}_i)$ of the start object v_i and the current embedding $\mathbf{x}_j$ of the end object v_j, so training is more efficient if at least one of the two embeddings is "clean" by directly encoding the label information. In this case, $\mathbf{x}_j$ is clean if v_j is labeled, but $\mathbf{G}_{p_{ij}}(\mathbf{x}_i)$ is not clean even if v_i is labeled. Therefore, we apply *reverse* path sampling, *i.e.*, we always sample paths from labeled objects, and use them in the reverse way, to make sure the end objects are always labeled.

We call this further improved model variant NEP-label. In our experiments, we observe that NEP-label leads to another $85\% - 96\%$ *shorter runtimes* until convergence and $2.3\% - 26\%$ *relative performance* gain compared with NEP-target.

Training with Two-Step Path Sampling, As we have discussed in Section III.D, one major advantage of NEP over existing SSL models on heterogeneous networks is the flexibility of considering arbitrary meta-paths and training the corresponding modular networks with path sampling based on uniform random walks on-the-fly, by leveraging the compositional nature of paths. However, since the modular networks composed for different paths have different neural architectures, it poses unique challenges for the efficient training of NEP by leveraging batch training, especially on GPUs.

We notice that, according to Eq. 9, paths sharing the same underlying meta-path should correspond to the same composed modular network. Therefore, although paths sampled by uniform random walks can be arbitrary, we can always group them into smaller batches according to their underlying meta-paths. However, path grouping itself is time consuming, and it leads to different group sizes, which is still not ideal for efficient batch training.

To address the challenges, we design a novel two-step path sampling approach for the efficient training of NEP, as depicted in Algorithm 1. Specifically, in order to sample a total number of Ω paths (*e.g.*, 100K), we firstly sample a smaller set of Γ paths (*e.g.*, 100). Then for each of these Γ paths, we find its underlying meta-path $\mathcal{M}$, and sample $B = \Omega/\Gamma$ paths (*e.g.*, 1K) that follow $\mathcal{M}$. Therefore, the particular modular network corresponding to $\mathcal{M}$ can be composed only once and efficiently trained with standard gradient back-propagation with the batch of B samples.

To sample random paths guided by particular meta-paths in Step 13, we follow the standard way in [20], [40]. However, different from them, our meta-paths are also sampled from the particular network, rather than given by domain experts or exhaustively enumerated. Assuming Ω is sufficiently large compared with B, the total Ω paths sampled by our two-step approach only differ in orders from any Ω paths sampled by the original approach, which corresponds to the special case with $B = 1$. Therefore, our path sampling approach is purely data-driven, and our model automatically learns their importance and complex functions regarding embedding propagation.

In Section IV, we show that our two-step path sampling approach can significantly reduce the runtimes of NEP, while it is also able to slightly boost its performance, due to more stable training and faster convergence.

Training Algorithm. Algorithm 1 gives an outline of our overall training process, which is based on NEP-label with two-step path sampling. In the inner loop starting from Line 6, it samples a path completely at random without the consideration of meta-path. Then it samples more instances under the same meta-paths. This strategy is crucial to eliminate the explicit consideration of a limited set of meta-paths, which makes NEP different from all existing SSL algorithms on heterogeneous networks. It is also crucial for leveraging batch-wise gradient backpropagation, which utilizes the power of dynamically composed modular networks. In this way, our model is data driven and able to consider any possible meta-paths underlying uniform random walks on heterogeneous networks.

Algorithm 1 Efficient Training of NEP

```
1: procedure NEPTRAIN
2:     Input: G, Ω, Γ, B(= Ω/Γ), max path length L
3:     for i ← 1 to Γ do
4:         Sample a source object v_s ∈ V_l
5:         p = (v_s)
6:         for j ← 1 to L do
7:             Sample (e_ij, v_j) on G from p[−1] and append to p
8:             if φ(v_j) ∈targeted object type then
9:                 break
10:            end if
11:        end for
12:        Find the underlying meta-path M of p
13:        Sample B paths under M from random labeled objects
14:        Take a gradient step to optimize J'
15:    end for
16: end procedure
```

Complexity Analysis. In terms of memory, the number of parameters in NEP is $O(N+L+T)$, where N is the number of targeted types of objects in the network, L and T are the number of classes and number of link types, respectively, which are independent of the network sizes. The $O(N)$ term is due to the embedding look-up table $\mathbf{E}$, which can be further reduced to a constant number if we replace it with an MLP given available object attributes. The $O(L)$ and $O(T)$ terms are due to the parameters in Θ_n of the predictive model and Θ_m in the link-wise neural network modules, respectively.

In terms of runtime, training NEP theoretically takes $O(\Omega L)$ time, where Ω is the number of sampled paths and L is the path length. It is the same as the state-of-the-art unsupervised heterogeneous network embedding algorithms [40], [20], while can be largely improved in practice based on the series of strategies we develop here.

F. Connections and Extensions

We show that NEP is a principled and powerful SSL framework by studying its connections to various existing graph-based SSL algorithms and promising extensions towards further improvements on modeling heterogeneous networks.

To better understand the mechanism of NEP, we analyze it in the well-studied context of graph signal processing [48], [49]. Specifically, we decompose NEP into three major components: *embedding*, *propagation*, and *prediction*, which can be mathematically formulated as $\mathcal{X} = \mathcal{F}(\mathcal{V})$, $\mathcal{X}' = \mathcal{G}(\mathcal{X})$, and $\hat{\mathcal{Y}} = \mathcal{Z}(\mathcal{X}')$, where $\mathcal{X}$ is the graph embedding with $\mathcal{F}$ as the embedding function, $\mathcal{X}'$ is the propagated embedding with $\mathcal{G}$ as the propagation function, and $\hat{\mathcal{Y}}$ is the label prediction with $\mathcal{Z}$ as the prediction function.

In NEP, we apply an embedding look-up table as $\mathcal{F}$ to directly capture the training labels $\mathcal{Y}$ on $\mathcal{V}_l$, while it is straightforward to implement $\mathcal{F}$ as an MLP to also incorporate object attributes $\mathcal{A}$ on $\mathcal{V}$, which we leave as future works. Such embedding allows us to further explore the complex interactions of labels and attributes among different types of objects. Our major technical contribution is then on the propagation function $\mathcal{G}$, which leverages modular networks to properly propagate object embeddings on different paths. Finally, due to the appropriate embedding and propagation functions, we are able to jointly learn a powerful parametric prediction function $\mathcal{Z}$ as an MLP based on very few samples.

Algorithm	$\mathcal{F}(\mathcal{V})$	$\mathcal{G}(\mathcal{X})$	$\mathcal{Z}(\mathcal{X}')$
LP [3]	$\mathcal{Y}$	$(I+\alpha L)^{-1}\mathcal{X}$	$\text{argmax}_j \mathcal{X}'_{ij}$
MR [50]	$\mathcal{A}$	$(I+\alpha L)^{-1}\mathcal{X}$	$\mathbf{W}\mathcal{X}' + \mathbf{b}$
Planetoid [11]	MLP($\mathcal{A}$)	$(I+\alpha L)^{-1}\mathcal{X}^*$	MLP($\mathcal{X}'$)
GCN [12]	MLP($\mathcal{A}$)	$(I-\tilde{L})^k\mathcal{X}$	MLP($\mathcal{X}'$)

TABLE I
DECOMPOSITION OF POPULAR GRAPH-BASED SSL ALGORITHMS.

In fact, we find that various existing graph-based SSL algorithms can well fit into this three-component paradigm, and they naturally boil down to certain special cases of NEP. As summarized in Table I, the classic LP algorithm [3] directly propagates object labels on the graph based on a deterministic auto-regressive function $(I+\alpha L)^{-1}$, where I is the identical matrix and L is the graph Laplacian matrix [51]. Prediction of LP is done by picking out the propagated labels with largest values. To jointly leverage object attributes and the underlying object links, MR [50] trains a parametric prediction model based on SVM with a graph Laplacian regularizer. To some extend, it can be viewed as propagating the object attributes on the graph. The recently proposed graph neural network models like Planetoid [11] and GCN [12] leverage object embedding to integrate object attributes, links and labels. Although the two works adopt quite distinct models, they essentially only differ in the implementations of the propagation function $\mathcal{G}$: Planetoid leverages random path sampling on networks to approximate the effect of graph Laplacian [17], whereas GCN sums up the embedding of neighboring objects in each of its k convolutional layers through a different smoothing function with $\tilde{L}$ as the normalized graph Laplacian with self-loops.

In concept, due to the expressiveness of neural networks, NEP can learn arbitrary propagation functions given proper training data, thus generalizing all of the above discussed algorithms. Moreover, since we notice that the propagation functions of most SSL algorithms on homogeneous networks are constructed to act as a deterministic low-pass filter that effectively encourages smoothness among neighboring objects, it is interesting to extend NEP by designing proper constraints on our modular networks to further construct learnable low-pass filters on heterogeneous networks.

IV. EXPERIMENTS

In this section, we comprehensively evaluate the performance of NEP for SSL over three massive real-world heterogeneous networks. The implementation of NEP is all public on GitHub[1].

A. Experimental Settings

Datasets. We describe the datasets we use for our experiments as follows with their statistics summarized in Table II.

1) **DBLP** We use the public Arnetminer dataset V8 collected by [52]. It contains four types of objects, *i.e.*, authors (A), papers (P), venues (V), and years (Y). The link types include authors writing papers, papers citing papers, papers published in venues, and papers published in years.
2) **YAGO** We use the public knowledge graph derived from Wikipedia, WordNet, and GeoNames [53]. There are seven types of objects in the network: person (P), location (L), organization (O), and *etc*, as well as twenty-four types of links.
3) **GitHub** We use an anonymous social network dataset derived from the GitHub community by DARPA. It contains three types of objects: repository (R), user (U) and organization (O), and six types of links, as depicted in Figure 2.

Dataset	#object	#edge	#class	%labeled
DBLP	4,925,160	44,931,742	4	0.081%
YAGO	545,792	3,517,663	15	0.166%
GitHub	2,078,030	61,332,330	16	0.197%
sub-DBLP	333,160	2,620,736	4	1.204%
sub-YAGO	15,672	3,57,312	15	37.219%
sub-GitHub	32,792	347,768	16	12.503%

TABLE II
THE STATISTICS OF DATASETS.

In order to compare with some state-of-art graph SSL algorithms that cannot scale up to large networks with millions of objects, we create a smaller sub-graph on each dataset by only keeping the labeled objects (both training and testing labels) and their direct neighbors. We also summarize their statistics in Table II.

Compared Algorithms. We compare NEP with the following graph-based SSL algorithms and network embedding algorithms:

- **LP** [3]: Classic graph-based SSL algorithm that propagates labels on homogeneous networks. To run LP on heterogeneous networks, we suppress the type information of all objects.
- **GHE** [13]: The state-of-the-art SSL algorithm on heterogeneous networks through path augmented and task guided embedding.
- **SemiHIN** [14]: Another recent SSL algorithm with promising results on heterogeneous networks by ensemble of meta-graph guided random walks.
- **ZooBP** [42]: Another recent SSL algorithm on heterogeneous networks by performing fast belief propagation.
- **Metapath2vec** [20]: The state-of-the-art heterogeneous network embedding algorithm through heterogeneous random walks and negative sampling.
- **ESim** [40]: Another recent heterogeneous network embedding algorithm with promising results through meta-path guided path sampling and noise-contrastive estimation.
- **Hin2vec** [41]: Another recent heterogeneous network embedding algorithm that exploits different types of links among nodes.

[1]https://github.com/JieyuZ2/NEP

Evaluation Protocols. We study the efficacy of all algorithms on the standard task of semi-supervised node classification. The labels are semantic classes of objects not directly captured by the networks. For DBLP, we use the manual labels of authors from four research areas, *i.e.*, database, data mining, machine learning and information retrieval provided by [18]. For YAGO, we extract top 15 locations by the edge "wasBornIn" as labels for the person objects and remove all "wasBornIn" links from the network. For GitHub, we manually select 16 high-quality tags of repositories as labels, such as security, machine learning, and database.

We randomly select 20% of labeled objects as testing data and evaluate all algorithms on them. For the SSL algorithms, (*i.e.*, LP, GHE, SemiHIN, ZooBP and NEP), we provide the rest 80% labeled objects as training data. For the unsupervised embedding algorithms, (*i.e.*, Metapath2vec, ESim and Hin2vec), we compute the embedding without training data. For all algorithms that outputs a network embedding (*i.e.*, Metapath2vec, ESim, Hin2vec and NEP), we train a subsequent MLP with the same architecture on the learned embeddings with the same 80% labeled training data to predict the object classes. Besides the standard classification accuracy, we also record the runtimes of all algorithms which are measured on a server with four GeForce GTX 1080 GPUs and a 12-core 2.2GHz CPU. For the sake of fairness, we run all algorithms with a single thread.

Parameter Settings. For all datasets, we set the batch size B to 1,000 and learning rate to 0.001. For different datasets, the number of sampled patterns Γ and maximum path length L are set differently as summarized in Table III. For all embedding algorithms, we set the embedding dimension to 128 for full graphs and 64 for sub-graphs. Other parameters of the baseline algorithms are set as the default values as suggested in the original works. For algorithms that require given sets of meta-paths (*i.e.*, GHE, SemiHin, Metapath2vec and ESim), since the schemas of our experimented heterogeneous networks are relatively simple, we compose and give them the commonly used meta-paths[2]. For NEP, we use single-layer MLPs (one fully-connected layer plus one Sigmoid activation layer) for all modules with the same size. We have also done a comprehensive study of the impacts of major hyper-parameters.

Dataset	DBLP	YAGO	GitHub	sub-DBLP	sub-YAGO	sub-GitHub
Γ	9000	7000	6000	6000	2000	2000
L	5	6	6	7	5	4

TABLE III
THE NUMBER OF SAMPLED PATTERNS AND MAXIMUM PATH LENGTHS FOR NEP ON DIFFERENT DATASETS.

Research Problems. Our experiments are designed to answer the following research questions:

- **Q1. Effectiveness** Given limited labeled data, how much does NEP improve over the state-of-the-art graph-based SSL and network embedding algorithms?
- **Q2. Efficiency** How efficient is NEP regarding the leverage of labeled data and computational resources?
- **Q3. Robustness** How robust is NEP regarding different settings of model hyper-parameters?

B. Q1. Effectiveness

We quantitatively evaluate NEP against all baselines on the standard node classification task. Table IV shows the performance of all

[2]DBLP: A-P-A, A-P-P-A, A-P-A-P-A, A-P-V-P-A, A-P-Y-P-A; YAGO:P-P, P-W-P, P-R-P, P-S-P, P-O-P, P-D-P, P-D-D-P, P-D-E-D-P; GitHub: R-R, R-U-R, R-U-U-R, R-U-R-U-R, R-U-O-U-R. The weights are all uniform.

algorithms on the six datasets. All algorithms are trained and tested with 10 runs on different randomly split labeled data to compute the average classification accuracy. The performance gain of NEP over baselines all passed the significant test with p-value 0.005. The performance of baselines varies across different datasets, while NEP is able to constantly outperform all of them with significant margins, demonstrating its supreme and general advantages.

Algorithm	sub-DBLP	sub-YAGO	sub-GitHub
LP	0.783±0.000	0.597±0.008	0.374±0.003
GHE	0.778±0.014	0.501±0.002	0.353±0.011
SemiHIN	0.787±0.000	0.630±0.000	0.306±0.000
ZooBP	0.680±0.000	0.382±0.000	0.312±0.000
Metapath2vec	0.851±0.003	0.604±0.003	0.384±0.003
ESim	0.824±0.005	0.563±0.004	0.342±0.006
Hin2vec	0.856±0.005	0.628±0.005	0.341±0.003
NEP-linear	0.885±0.003	0.648±0.003	0.400±0.180
NEP	**0.888±0.005**	**0.651±0.002**	**0.425±0.007**
Algorithm	**DBLP**	**YAGO**	**GitHub**
LP	0.811±0.033	0.612±0.004	0.340±0.006
GHE	0.759±0.048	0.447±0.020	0.351±0.019
SemiHIN	0.724±0.000	0.457±0.000	0.348±0.000
ZooBP	0.610±0.000	0.561±0.000	0.302±0.000
Metapath2vec	0.790±0.005	0.590±0.005	0.320±0.007
ESim	0.647±0.010	0.607±0.004	0.305±0.003
Hin2vec	0.836±0.001	0.609±0.004	0.338±0.006
NEP-linear	0.865±0.003	0.629±0.002	0.384±0.002
NEP	**0.880±0.006**	**0.634±0.005**	**0.392±0.011**

TABLE IV
OVERALL EFFECTIVENESS OF COMPARED ALGORITHMS.

Taking a closer look at the scores, we observe that NEP is much better than baselines on the sub-graphs of YAGO and GitHub, where the graphs have relatively complex links but small sizes. For example, in GitHub, a repository and a user can have "watched by" and "created by" links, while in YAGO, a person and a location can have "lives in", "died in", "is citizen of" and other types of links. On these graphs, NEP easily benefits from its capability of distinguishing and leveraging different types of direct interactions through the individual neural network modules. However, when evaluated on full graphs, the performance gain of NEP is larger on DBLP. The fact is, since direct interactions are simpler in DBLP, where only a single type of link exists between any pair of objects, higher-order interactions matter more. For example, a path of "A-P-V-P-A" exactly captures the pairs of authors within the same research communities. The full graphs, compared with the sub-graphs, can provide much more instances of such longer paths, and NEP effectively captures such particular higher-order interactions through learning the dynamically composed modular networks.

The advantage of NEP mainly roots in two perspectives: the capability of modeling the compositional nature of meta-paths and the flexibility of non-linear propagation functions. To verify this, we also implement a linear version of NEP by simply removing all non-linear activation functions. As we can clearly see, the performance slightly drops after removing the nonlinearity in a consistent way.

C. Q2. Efficiency

In this subsection, we study the efficiency of NEP regarding the leverage of both labeled data and computational resources.

One of the major motivations of NEP is to leverage limited labeled data, so as to alleviate the deficiency of deep learning models when training data are hard to get. Therefore, we are interested to see how NEP performs when different amounts of training data are available. To this end, we change the amount of training data from 10% to 90% of all labeled data, while the rest 10% labeled data are held out as testing data. We repeat the same process but split the data randomly for 10 times to take the average scores.

As we can observe in Figure 3, NEP can quickly capture the simple semantics in DBLP and reaches stable performance given only $10\% - 20\%$ of the labeled data. Although YAGO and GitHub appear to be more complex and require relatively more training data, NEP maintains the best performances compared with all baselines. Such results clearly demonstrates the efficiency of NEP in leveraging limited labeled data.

Another major advantage of neural network models is that they can usually be efficiently trained on powerful computation resources like GPUs with well-developed optimization methods like batch-wise gradient backpropagation. Particularly for NEP, as we have discussed in Section III.E, since our modular networks are dynamically composed according to randomly sampled paths, we have developed a novel training strategy based on two-step path sampling to fully leverage the computational resources and standard optimization methods. Here we closely study its effectiveness.

Figure 4 shows how the strategy of training with two-step path sampling influences the performance and runtime of NEP regarding different settings on the three datasets. To present a comprehensive study, we set the total number of sampled paths (Ω) to 100K, 500K, and 1M, respectively, and then simultaneously vary the number of patterns (Γ) and the number of paths per pattern (B). As we can observe from the results, when $B = 1$, which equals to no usage of two-step sampling, the runtimes are quite high; as we increase B to 10 and 10^2, the runtimes rapidly drop, while the performances are not influenced much. Sometimes the performance actually increases, probably due to better convergence of the loss with batch training. Setting B to too large values like 10^3 does hurt the performance, but in practice, we can safely avoid it by simply setting B to an appropriate value like 10^2, which leads to a satisfactory trade-off between effectiveness and efficiency across different datasets.

D. Q3. Robustness

We comprehensively study the robustness of NEP regarding different hyper-parameter settings.

We firstly look at the path length L. As shown in Table V, the performance of NEP regarding different path lengths does not differ significantly as we vary L from 2 to 7. This is because shorter paths are usually more useful, and when L is large, Algorithm 1 often automatically stops the sampling process at Line 9 upon reaching targeted types of objects before the actual path length reaches L. Therefore, in practice, the rule-of-thumb is to simply set L to larger values like 5-6.

Then we look at the embedding size K, by comparing NEP with GHE, Metapath2vec, ESim and Hin2vec, which also compute object embeddings. As shown in Figure 5, too small embedding sizes often lead to poor performance. As the embedding size grows, NEP quickly reaches the peak performance. It also maintains the best performance without overfitting the data as the embedding size further grows.

Finally we look at the number of total sampled paths Ω, by comparing NEP with Metapath2vec, ESim and Hin2vec, which are also trained with path sampling. As shown in Figure 6, the improvement of NEP over compared baselines is more significant given fewer sampled paths, indicating the power of NEP to rapidly capture useful information in the networks.

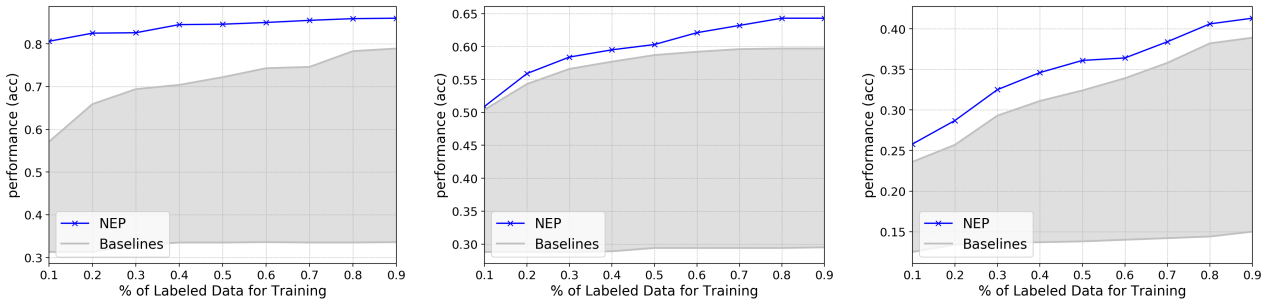

Fig. 3. **Efficiency on leveraging limited labeled data (from left to right: DBLP, YAGO, GitHub).**

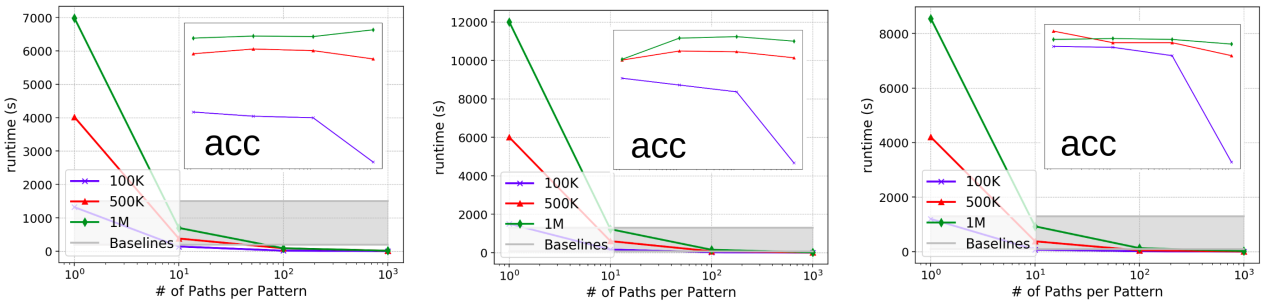

Fig. 4. **Efficiency on leveraging computational resources (from left to right: DBLP, YAGO, GitHub).**

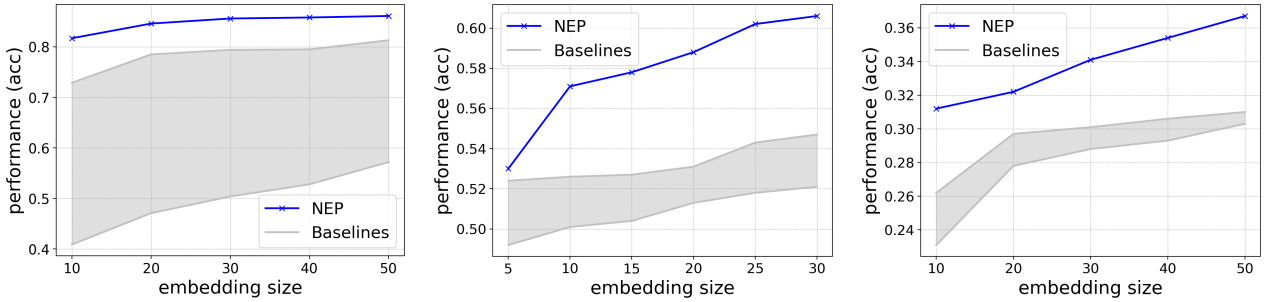

Fig. 5. **Robustness of NEP regarding embedding sizes (from left to right: DBLP, YAGO, GitHub).**

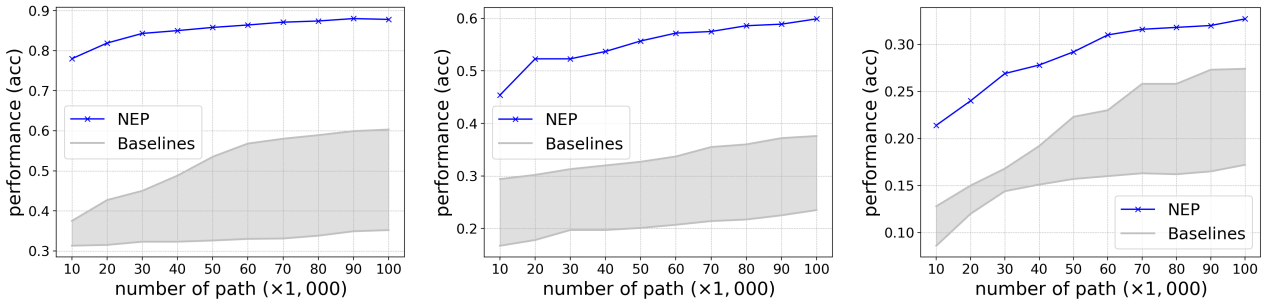

Fig. 6. **Robustness of NEP regarding numbers of sampled paths (from left to right: DBLP, YAGO, GitHub).**

Dataset	2	3	4	5	6	7
DBLP	0.750	0.872	0.869	0.880	0.873	0.875
YAGO	0.631	0.631	0.633	0.629	0.634	0.631
GitHub	0.381	0.374	0.378	0.386	0.392	0.379
sub-DBLP	0.763	0.875	0.880	0.886	0.881	0.888
sub-YAGO	0.646	0.641	0.641	0.651	0.637	0.648
sub-GitHub	0.425	0.412	0.414	0.412	0.405	0.407

TABLE V
ROBUSTNESS OF NEP REGARDING PATH LENGTHS.

V. CONCLUSIONS

In this work, we develop NEP (*Neural Embedding Propagation*), for semi-supervised learning over heterogeneous networks. NEP is a powerful yet efficient neural framework that coherently combines an object encoder and a modular network to model the complex interactions among multi-typed multi-relational objects in heterogeneous networks. Unlike existing heterogeneous network models, NEP does not assume a given set of useful meta-paths, but rather dynamically composes and estimates the different importance and functions of arbitrary meta-paths regarding embedding propagation on-the-fly. At the same time, the model is easy to learn, since the parameters modeling each type of links are shared across all underlying meta-paths. For future works, it is straightforward to extend NEP to various object attributes and enable fully unsupervised training for by recovering different types of links.

ACKNOWLEDGEMENTS

Research was sponsored in part by U.S. Army Research Lab. under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA), DARPA under Agreement No. W911NF-17-C-0099, National Science Foundation IIS 16-18481, IIS 17-04532, and IIS-17-41317, DTRA HDTRA11810026, and grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov).

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[3] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," 2002.
[4] X. Zhu, Z. Ghahramani, J. Lafferty *et al.*, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, vol. 3, 2003.
[5] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *NIPS*, 2016, pp. 3630–3638.
[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICLR*, 2017.
[7] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017.
[8] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly, "Video suggestion and discovery for youtube: taking random walks through the view graph," in *WWW*, 2008.
[9] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.
[10] B. Lin, J. Yang, X. He, and J. Ye, "Geodesic distance function learning via heat flow on vector fields," in *ICML*, 2014, pp. 145–153.
[11] Z. Yang, W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, 2016.
[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[13] T. Chen and Y. Sun, "Task-guided and path-augmented heterogeneous network embedding for author identification," in *WSDM*, 2017.
[14] H. Jiang, Y. Song, C. Wang, M. Zhang, and Y. Sun, "Semi-supervised learning over heterogeneous information networks by ensemble of meta-graph guided random walks," in *AAAI*, 2017, pp. 1944–1950.
[15] C. Yang, Y. Feng, P. Li, Y. Shi, and J. Han, "Meta-graph based hin spectral embedding: methods, analyses, and insights," in *ICDM*, 2018.
[16] C. Yang, M. Liu, F. He, X. Zhang, J. Peng, and J. Han, "Similarity modeling on heterogeneous networks via automatic path discovery," in *ECML-PKDD*, 2018.
[17] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han, "Bridging collaborative filtering and semi-supervised learning: a neural approach for poi recommendation," in *KDD*, 2017, pp. 1245–1254.
[18] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *VLDB*, vol. 4, no. 11, pp. 992–1003, 2011.
[19] Y. Shi, P.-W. Chan, H. Zhuang, H. Gui, and J. Han, "Prep: Path-based relevance from a probabilistic perspective in heterogeneous information networks," in *KDD*, 2017, pp. 425–434.
[20] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017.
[21] M. Ji, Y. Sun, M. Danilevsky, J. Han, and J. Gao, "Graph regularized transductive classification on heterogeneous information networks," in *ECML-PKDD*, 2010, pp. 570–586.
[22] C. Luo, R. Guan, Z. Wang, and C. Lin, "Hetpathmine: A novel transductive classification algorithm on heterogeneous information networks," in *ECIR*, 2014, pp. 210–221.
[23] C. Wang, Y. Song, H. Li, M. Zhang, and J. Han, "Knowsim: A document similarity measure on structured heterogeneous information networks," in *ICDM*, 2015, pp. 1015–1020.
[24] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *CVPR*, 2016, pp. 39–48.
[25] ——, "Learning to compose neural networks for question answering," in *NAACL*, 2016, pp. 1545–1554.
[26] G. Jeh and J. Widom, "Scaling personalized web search," in *WWW*, 2003, pp. 271–279.
[27] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.
[28] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.
[29] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.
[30] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *JMLR*, 2011.
[31] Y. Sun and J. Han, "Mining heterogeneous information networks: principles and methodologies," *SLKDD*, vol. 3, no. 2, pp. 1–159, 2012.
[32] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. Chang, M. Wu, and J. Ying, "Semantic proximity search on heterogeneous graph by proximity embedding." in *AAAI*, 2017, pp. 154–160.
[33] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *KDD*, 2017, pp. 1507–1515.
[34] Y. Sun, B. Norick, J. Han, X. Yan, P. S. Yu, and X. Yu, "Integrating meta-path selection with user-guided object clustering in heterogeneous information networks," *TKDD*, vol. 7, no. 3, p. 11, 2013.
[35] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee, "Meta-graph based recommendation fusion over heterogeneous information networks," in *KDD*, 2017, pp. 635–644.
[36] H. Zhuang, J. Zhang, G. Brova, J. Tang, H. Cam, X. Yan, and J. Han, "Mining query-based subnetwork outliers in heterogeneous information networks," in *ICDM*, 2014, pp. 1127–1132.
[37] M. Wan, Y. Ouyang, L. Kaplan, and J. Han, "Graph regularized meta-path based transductive regression in heterogeneous information network," in *SDM*, 2015, pp. 918–926.
[38] X. Li, B. Kao, Y. Zheng, and Z. Huang, "On transductive classification in heterogeneous information networks," in *CIKM*, 2016, pp. 811–820.
[39] Y. Fang, W. Lin, V. W. Zheng, M. Wu, K. Chang, and X.-L. Li, "Semantic proximity search on graphs with metagraph-based learning," in *ICDE*, 2016, pp. 277–288.
[40] J. Shang, M. Qu, J. Liu, L. M. Kaplan, J. Han, and J. Peng, "Meta-path guided embedding for similarity search in large-scale heterogeneous information networks," *arXiv preprint arXiv:1610.09769*, 2016.
[41] T.-y. Fu, W.-C. Lee, and Z. Lei, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *CIKM*, 2017, pp. 1797–1806.
[42] D. Eswaran, S. Günnemann, C. Faloutsos, D. Makhija, and M. Kumar, "Zoobp: Belief propagation for heterogeneous networks," *VLDB*, vol. 10, no. 5, pp. 625–636, 2017.
[43] C. Yang, L. Zhong, L.-J. Li, and L. Jie, "Bi-directional joint inference for user links and attributes on large social graphs," in *WWW*, 2017, pp. 564–573.
[44] F. Serafino, G. Pio, and M. Ceci, "Ensemble learning for multi-type classification in heterogeneous networks," *TKDE*, vol. 30, no. 12, pp. 2326–2339, 2018.
[45] C. Yang, C. Zhang, X. Chen, J. Ye, and J. Han, "Did you enjoy the ride: Understanding passenger experience via heterogeneous network embedding," in *ICDE*, 2018.
[46] Y. Shi, X. He, N. Zhang, C. Yang, and J. Han, "User-guided clustering in heterogeneous information networks via motif-based comprehensive transcription."
[47] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *nature*, vol. 393, no. 6684, p. 440, 1998.
[48] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *ISPM*, vol. 30, no. 3, pp. 83–98, 2013.
[49] B. Girault, P. Gonçalves, E. Fleury, and A. S. Mor, "Semi-supervised learning for graph to signal mapping: A graph signal wiener filter interpretation," in *ICASSP*, 2014, pp. 1115–1119.
[50] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *JMLR*, vol. 7, no. Nov, pp. 2399–2434, 2006.
[51] D. A. Spielman, "Spectral graph theory and its applications," in *FOCS*, 2007, pp. 29–38.
[52] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *KDD*, 2008.
[53] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *WWW*, 2007, pp. 697–706.