PyDHNet: A Python Library for Dynamic Heterogeneous Network Representation Learning and Evaluation

Hoang Nguyen  
Radin Hamidi Rad  
Ebrahim Bagheri

hoang.cam.nguyen@ryerson.ca  
radin@ryerson.ca  
bagheri@ryerson.ca

Toronto Metropolitan University, ON, Canada  
Toronto Metropolitan University, ON, Canada  
Toronto Metropolitan University, ON, Canada

ABSTRACT

Network representation learning and its applications have received increasing attention. Due to their various application areas, many research groups have developed a diverse range of software tools and techniques to learn representation for different types of networks. However, to the best of our knowledge, there are limited works that support representation learning for dynamic heterogeneous networks. The work presented in this demonstration paper attempts to fill the gap in this space by developing and publicly releasing an open-source Python library known as PyDHNet, a Python Library for Dynamic Heterogeneous Network Representation Learning and Evaluation. PyDHNet consists of two main components: dynamic heterogeneous network representation learning and task-specific evaluation. In our paper, we demonstrate that PyDHNet has an extensible architecture, is easy to install (through PIP) and use, and integrates quite seamlessly with other Python libraries. We also show that the implementation for PyDHNet is efficient and enjoys a competitive execution time.

CCS CONCEPTS

• Computing methodologies → Learning latent representations;  
• Mathematics of computing → Graph algorithms;  
• Information systems → Retrieval models and ranking.

KEYWORDS

Network Representation Learning, Dynamic Heterogeneous Network, Network Application

1 INTRODUCTION

Network representation learning is the task of mapping a graphical network structure into a lower dimensional embedding vector space. The embedded vectors should be compact to practically efficient while still capturing and maintaining network properties for various downstream network tasks. With the increasing importance of the network representation learning task, several open-source libraries and tools have been progressively developed and shared in the community [7, 11, 12, 17, 22, 25]. We summarize some of the main libraries for this task in Table 1. As shown in the table, the majority of existing libraries are designed for learning network representation for static networks. In other words, they consider the graph to be a single snapshot and do not support for evolving graphs over time. Furthermore, these libraries often overlook the need for more complex network representations by only allowing for homogeneous node and edge types in the network.

From the existing libraries, DynamicGEM and CTGCN are among the first to allow representations to be learnt for dynamically evolving graphs. However, both of these libraries are limited by only supporting homogeneous networks. In contrast, OpenAttHetRL and Space4HGNN are the only two libraries that support for network heterogeneity; however, they do not support for network dynamism. The work in this paper illustrates our work on an open-source python-based network representation library, called PyDHNet, which supports both network heterogeneity and network dynamism. Mentioned features are not currently supported by any other available libraries. We believe that PyDHNet is an extremely valuable contribution to the community in many application domains, such as social networks, biological dataset and traffic monitoring data, to name just a few, deal with graphs that rely on efficient representations through dynamic heterogeneous graphs.

The purpose of this demonstration can be enumerated as follows:

1) we will introduce PyDHNet, which allows users to both efficiently and effectively learn network representations for dynamic heterogeneous networks;

2) we will show how the structure of PyDHNet has been designed based on software engineering principles make it easily extensible for adding new features. Our developed software library is modularized and hence easy to maintain and work with;

3) we will demonstrate the PyDHNet consists of a standalone evaluation library that supports for the testing the performance of the learnt representations on different downstream tasks such as node classification and link prediction, which makes it ideal for comparative analysis and replicable research.

The rest of this demonstration paper is organized as follows: We will first provide an overview of our representation learning technique and how it differentiates itself from other existing work in Section 2. Next, the software architecture of PyDHNet is briefly...
This approach allows modelling temporal dependencies without having to consider the subgraph structure of the node when learning network representations. A static network at time $t$ is formalized as $G_t = (V_t, E_t)$, where $V_t$ is the node set and $E_t$ is the edge set. In order to deal with heterogeneous networks, at each timestamp $t$, we formalize $G_t$ as a static heterogeneous network, which consists of multiple node types. As such and in each timestamp $t$, our dynamic network representation learning method aims to learn latent representations for each node in the network $G$ in such a way that it preserves both the properties of the node in previous and current heterogeneous network snapshots as well as its evolutionary behavior up to time $t$.

There are two main approaches to deal with heterogeneity at each network snapshot: (1) one approach adopts a type-specific mapping function to aggregate the information from the directed neighbors (e.g., HGAT [22], HGT [11]); (2) the other approach captures higher-order proximity by employing meta-path neighborhood aggregation (e.g., MAGNN [7], HeteGNNs [12]). The implementation of such methods can be found in the latest libraries such as OpenNE and Space4HGNN. However, none of these libraries consider the subgraph structure of the node when learning network representations. The network learning method used in our library utilizes a meta-path neighborhood aggregation approach to capture higher-order proximity information while at the same time considering the subgraph structures around each node. As such, our approach for capturing heterogeneity is unique as it captures both local subgraph structures surrounding each node as well as higher-order proximity information.

For the sake of encoding the temporal nature of the dynamic graph, a natural approach is to use time series encoders since they support node addition/deletion and link update [19]. However, the research community has recently found that the use of positional encodings is a more effective approach compared to recurrent neural networks especially due to its computational efficiency [1, 18].

This approach allows modelling temporal dependencies without the need of feeding the sequential data recursively, which supports parallel computation. Our library uses Transformer-like positional encoding [8] in order to capture the temporal order of network snapshots. In our library, the positional encoding first encodes the order of the network snapshots regarding the position of their timestamp in the whole time series of the network. The position encoding vectors are then combined with the input vectors to model the sequence of the input representation.

Based on these techniques, our representation learning technique can capture both the heterogeneity as well as the evolution of dynamic heterogeneous networks, and allows users to have richer network representations with potentially better performance on downstream tasks.

### 3 PYDHNET STRUCTURE

In this section, we present the design of the PyDHNet library and show that it is quite easy to use and extend it by other users. The overall structure of PyDHNet is shown in Figure 1. As shown in this figure, the library consists of two main components, Network Representation Learning designed for developing embedding representation, and Network Evaluation for assessing the quality of the learnt representations on downstream tasks. The former component consists of three main tasks, namely (1) data preparation, (2) model training, and (3) embedding generation. The latter component supports the evaluation of the embedding representations based on downstream tasks such as node classification and link prediction. This would make PyDHNet quite suitable to serve as a baseline for many domain-specific tasks such as expertise prediction (node classification) [13, 21], and product recommendation (link prediction) [2, 5], to name a few.

The network representation learning component of the PyDHNet library is built upon PyTorch Geometric (PyG) [6] and Pytorch Lightning [4] while the network evaluation component is built to be compatible with Scikit-learn [16]. The mentioned libraries are widely used in the Machine Learning (ML) community and have well-adopted standards for data input, model export and evaluation. As a result, PyDHNet offers the following very desirable characteristics: (1) it is quite easy to use as it can be installed simply through the ‘`PIP`’ command; (2) it relies on dependencies that are already well-managed through the Pytorch libraries; and is easy to learn as it uses Pytorch standards for development and Scikit-learn standards for model training, export and evaluation.

#### 3.1 Network Representation Learning Component

We have implemented the network representation learning component of PyDHNet in an object-oriented programming (OOP) style with the back-end based on well-known set of Python libraries, i.e.,

<table>
<thead>
<tr>
<th>Tool</th>
<th>First release</th>
<th>Library Features</th>
<th>Environment</th>
<th>Heterogeneity</th>
<th>Node properties</th>
<th>Evaluation</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenNE [20]</td>
<td>2018</td>
<td>Static</td>
<td>Both</td>
<td>No</td>
<td>No</td>
<td><a href="https://github.com/thunlp/OpenNE">https://github.com/thunlp/OpenNE</a></td>
<td></td>
</tr>
<tr>
<td>Space4HGNN [27]</td>
<td>2022</td>
<td>Static</td>
<td>Heterogeneous</td>
<td>No</td>
<td>Yes</td>
<td><a href="https://github.com/BUPT-GAMMA/OpenHGNN">https://github.com/BUPT-GAMMA/OpenHGNN</a></td>
<td></td>
</tr>
<tr>
<td>PyDHNet</td>
<td>2022</td>
<td>Dynamic</td>
<td>Heterogeneous</td>
<td>No</td>
<td>Yes</td>
<td><a href="https://github.com/hoangntc/PyDHNet">https://github.com/hoangntc/PyDHNet</a></td>
<td></td>
</tr>
</tbody>
</table>
Pytorch, Pytorch Lightning and Pytorch Geometric. As a result, this allows users in this space to easily pick up our library and use it without much learning barriers for their specific downstream task. **Data Preparation** In this task, the users are able to convert the format of the input dataset to the general standard format used for graph. We support a similar graph structure input to PyG except that our graph snapshots require temporal labels as well to support for dynamicity. We note that given the fact that we deal with dynamic networks, each graph consists of multiple snapshots. Each network snapshot is considered to be a static heterogeneous network and wrapped up as Dataset object from PyG. The network features at the node level are then extracted using the PyG library which enables the standardization of the dimension of node features used in the later stages. **Model Training** In the model training task, PyDHNet provides support for data loading, model training and model evaluation. Three sub-modules, namely DataModule, Module and Trainer, are responsible for the mentioned tasks, respectively. The DataModule handles the construction and the flow of the large network data during training process. In order to support for the streaming nature of dynamic networks, this module loads the data in mini batches with the temporal and structural information related to a node such as the node features, the node type and the structural subgraphs it belongs to in each timestamp. The Module handles the model generation task and receives the output of the DataModule as its input and generates the appropriate embeddings for the targeted network. The optimization during the learning process is defined under the Trainer. The Trainer provides commonly-used utility functions such as early stopping, regularization and checkpoint tracking. The configuration of all of the mentioned sub-modules such as the data batch size, the learning rate, the dropout probability or the loss criteria are stored in a .json file or can be inputted via a dictionary.

These sub-modules are inherited from the LightningDataModule, LightningModule and Trainer classes of the Pytorch Lightning library, which allows the users to easily and quickly implement or customize their own learning process.

**Network Generation** The objective of this module is to generate the embedding vectors for all the nodes in the network that can later be used for downstream tasks, e.g., node classification or link prediction. We note that PyDHNet generates node representations for each node up to each timestamp that it has observed. Therefore, the node representation will be updated at each timestamp.

### 3.2 Network Evaluation Component

Given a specific task, the network evaluation component receives the embedding vectors of all nodes or a set of node pairs as the input and employs a machine learning predictor (e.g., Logistic Regression) or a statistical measures (e.g., sigmoid function) to predict the score of the targeted output. Finally, it calculates a set of widely-used evaluation metrics, including but not limited to, accuracy, recall and precision, based on comparison between the ground truth and the predicted results.

### 4 PYDHNET DEMONSTRATION

In this section, we practically demonstrate how PyDHNet can be easily integrated into any Python program using its easy to use API. The PyDHNet tool requires four input files:

- **node_types** gives the detail of the type of all nodes in the network.
- **temporal_edge_list** tracks the existence of the edges corresponding to different timestamps.
- **temporal_subgraphs** includes the information of all the structural subgraphs in the network.
- **data** stores the information related to the nodes including the node index, the subgraph indices which it belongs to and whether the node is used for training/validation/testing.

To sample the structural subgraphs corresponding to each node at each time step, we provide a TemporalSubgraphSampler, which provides the implementation for some built-in sampling functions.
such as DiffusionSampler and RandomWalkSampler. An example on how to use the subgraph sampler is depicted in Figure 2.

```python
from PyDHNet.subgraph_sampler import TemporalSubgraphSampler

sampler = TemporalSubgraphSampler()
node_path =
edge_path =
sampled_node_id =
max_time =
num_nodes =
out_path =

sampler = TemporalSubgraphSampler()

sampler = TemporalSubgraphSampler()

sampler = TemporalSubgraphSampler()

Figure 2: Code Snippet for Temporal Subgraph Sampler.
```

Figure 3 demonstrates the implementation for the network representation learning component. The main component of PyDHNet can be initialized either by a file path which points to a configuration file or a direct configuration dictionary. Once the instance is created, the users are able to run the whole pipeline of the process in one full execution or in separate steps, both of which are supported.

```python
from PyDHNet import PyDHNet

# Create initialization
python = PyDHNet(config_path="/pydhnet/config/dblp.json")

# or
config_dict =

python = PyDHNet(config_dict=config_dict)

# Manual running
# 1. Data preprocessing
python = preprocess_data()

# 2. MetaModule, ModuleModule, Trainer initialization
data_module, model_module, trainer = python.initialize()
python.train(data_module, model_module, trainer)

# 3. Embedding generation
restore_model_dir = str(python.config['checkpoint_dir'])
restore_model_name = "name-ckpt"
output_dir = str(PYDHNet_PATH / 'output')
python.generate_embedding()

# Full pipeline running
python = run_pipeline()
```

Figure 3: Code Snippets for the three tasks offered by the Network Representation Learning component.

After obtaining the learnt model and inferring the network embedding, it is possible to easily evaluate the learnt representations using the PyDHNet network evaluation component. For clarity, we demonstrate how evaluation based on two downstream tasks can be performed in Figure 4. It is worth noting that evaluation is an independent component of our library, which can be used in conjunction with other network representation libraries as well. The evaluation component only requires the embeddings generated by any network representation learning libraries (often referred to as the node/node pair features), the label (the class of the node w.r.t. the task) and the sample dataset indicator (whether the sample is for training/validation/testing) in order to produce the evaluation metrics.

5 EXECUTION PERFORMANCE

It is also important to comparatively analyze the execution time of PyDHNet and the latest graph representation libraries, namely CTDGCN and OpenHINE. While other libraries do not simultaneously support for dynamicity and heterogeneity, we show that PyDHNet still shows competitive execution time. In order to compare the performance of different libraries, we run experiments over the DBLP dataset for the node classification task, which aims to predict the research area of an author as suggested in [21, 24]. Based on the dataset, we construct a network with 5,343 papers, 1,475 authors, 1,951 terms and 5 venues forming 55,310 interactions between the nodes in total. There are 8 timestamps in the dataset. For tools that support only static graphs, we consolidate nodes and edges from all timestamps into one graph. For methods that support only homogeneous nodes, we consider all nodes to be of the same type as suggested by [23, 26].

In all the experiments, the hidden dimension is set to 128, the batch size is 64 and the number of iterations is 50. We report execution time on a machine with one NVIDIA GeForce RTX 3090.

<table>
<thead>
<tr>
<th>Model</th>
<th>Tool</th>
<th>Network</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN [14]</td>
<td>CTDGCN</td>
<td>Static Homogeneous Network</td>
<td>6 mins</td>
</tr>
<tr>
<td>MetaGraph2Vec [25]</td>
<td>OpenHINE</td>
<td>Static Heterogeneous Network</td>
<td>35 mins</td>
</tr>
<tr>
<td>CTDGCN [15]</td>
<td>CTDGCN</td>
<td>Dynamic Homogeneous Network</td>
<td>2 hours 25 mins</td>
</tr>
<tr>
<td>PyDHNet</td>
<td>PyDHNet</td>
<td>Dynamic Heterogeneous Network</td>
<td>1 hours 45 mins</td>
</tr>
</tbody>
</table>

Table 2: The execution time of some common open-source tools for network representation learning.

The execution times are reported in Table 2 for the different methods. As seen in the table, methods that are designed for homogeneous networks are faster than those for heterogeneous networks (e.g., compare GCN with MetaGraph2Vec), and dynamic networks are lengthier to process compared to static networks (compare CTDGCN with MetaGraph2Vec). We note however PyDHNet has an efficient implementation where its execution time for a dynamic 'heterogeneous' network is even faster than other strong implementations for dynamic 'homogeneous' networks.

6 CONCLUDING REMARKS

In this demonstration paper, we have introduced the open-source python-based PyDHNet library that supports the task of learning embedding representations for Dynamic Heterogenous Network Representation Learning. We have shown that PyDHNet provides support for both dynamic (temporal) and heterogeneous networks, which are currently not supported by existing tools. Furthermore, we explained how PyDHNet is well-structured, extensible and easy to use, which makes it suitable for both practical use and further development. In addition, we have empirically shown the implementation of PyDHNet is quite efficient as it even has a faster performance compared to existing state of the art dynamic homogeneous network representation learning tools.
REFERENCES


