CrossWalk: Learning Representations Across Multiple Graphs

Devin de Hueck
Red Hat Inc.
Boston University
ddehueck@redhat.com

ABSTRACT

Graphs provide a wonderful way to interpret the complex relationships found in data. However, this data may be spread across multiple domains, e.g. a python package can be characterized by a language graph from documentation, a python code graph, and a dependency graph. This same problem emerges across a wide variety of data-hungry industries where no single data domain is sufficient to represent the entity being sought after i.e. a python package, a user in social media network, or a web page. In this paper we help address this problem with a two-fold contribution: (1) We introduce the PyPI Graph Dataset that includes three graph data domains. (2) We benchmark this dataset with a novel unsupervised xsmethod that extends random walk models to cross multiple domains that we coin CrossWalk.

CCS CONCEPTS

• **Information systems → Document representation**; • **Computing methodologies →** *Unsupervised learning*; *Transfer learning*.

ACM Reference Format:
Devin de Hueck. 2018. CrossWalk: Learning Representations Across Multiple Graphs. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

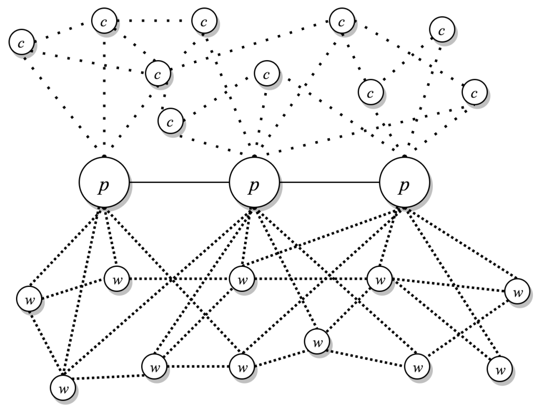

Figure 1: CrossWalk's Graphical Interpretation of the PyPI dataset. Notice the three distinct graphs, i.e domains, and how each relates to a python package.

1 INTRODUCTION

One of deep learning's most fundamental advances is the ability to learn meaningful representations from raw data. Recently there has been greater emphasis on learning representations from graph data. These methods are concerned with embedding nodes, subgraphs, or entire graphs into some lower dimensional latent space.

Recent work in this area seems to focus on the idea of graph convolutional networks (GCNs) that aggregate local features to build node representations [4, 5, 10]. While these methods have been shown to perform very well on a variety of tasks they only go so far. In the GCN paradigm a graph G has a set of vertices or nodes V, a set of edges E, and each node $v_i \in V$ has a feature matrix $\mathbf{X_i}$.

In the case that each node has graphical features one must first learn a vector representation to add them to each $\mathbf{X_i}$. Take the example of a dataset of python packages. Each package has textual data from documentation which can be interpreted as a language graph [9], a code graph, and a place in a larger dependency graph. In the GCN paradigm one would first retrieve document embeddings for the language graph and code graph then construct a feature matrix of these embeddings for each node. The datasets referenced in current GCN literature provide these node feature matrices, however, this forces researchers to work in this one interpretation of learning graph representations - potentially missing out on a better one.

In this paper, we provide a new interpretation for learning on graphical data with graphical features. Looking through a new lens, let G_s be a supergraph with the set of vertices V_s being the set of entities we would like to learn a representation for. Instead of a feature matrix $\mathbf{X_i}$ for each node in V, V_s is now connected to a set of graphs: $S = \{G_0 ... G_n\}$. Each $G_i \in S$ can be of two kinds - global or local. A global graph G_g has a set of edges E_g between the global supergraph nodes V_s (the entities we seek to represent) so there is no need for a new set of nodes. However, a local graph G_l has a new set of nodes V_l and - to relate the supergraph's nodes to this local graph - E_l also includes edges between V_l and V_s in addition to edges between V_l.

In the case of our python package example, each node in V_s corresponds to a python package in our dataset and S contains three graphs: a language graph, a code graph, and a dependency graph. Of these three, the dependency graph is the only global graph as the python packages are connected between themselves.

In the case of the local language graph we have a new set of nodes - the words. By connecting each package node in V_s to the word nodes we allow for the creation of a document vector as shown in [6]. We follow the same idea for the code graph where each local node is a syntax token (perhaps taken from an abstract syntax tree [2]).

The interpretation introduced in the previous two paragraphs is the first contribution of the CrossWalk model. The second is the extension of random walk methods to learn the representations being sought after. Applying CrossWalk to the python package example we can learn a single representation which encodes the information in each graph of S. Exactly how this is done is left to section 4. To validate this claim we require a dataset with all raw information available. Unfortunately, using the datasets consistently referenced in the GCN literature is not possible as they follow the feature matrix paradigm outlined above. In response, we have created our own dataset of python packages - the PyPI Dataset. The PyPI dataset has 200k python packages with the same three graphs in our running example: (1) a text graph (2) a code graph and (3) a dependency graph. We find that applying CrossWalk to the PyPI dataset validates our novel interpretation and random walk method so that we can learn from graph data with raw graphical features.

2 RELATED WORK

Graph Convolutional Networks: This class of networks aggregate local features generalizing the idea of Convolutional Neural Networks commonly used in the field of Computer Vision.

Random Walk Methods: DeepWalk [9], node2vec [3], sub2vec [1] all extend Mikolov et al.'s skip-gram language model [7] to arbitrary graphs with the use of random walks. A random walk begins at a randomly chosen node v, then another node is randomly chosen from v's 1-degree neighbors and so forth until a given length is met. As pointed out in [9], a sentence is analogous to a random walk through a language graph so the skip-gram language model [7] can be applied to a corpus of random walks to learn node representations. DeepWalk and word2vec both seek to find the parameters θ that maximize the following:

$$\arg\max_{\theta} \prod_{v_c, v_t}^{W} p(v_t | v_c; \theta) \tag{1}$$

Where W is all context node v_c and target node v_t pairs over all windows in the random walk corpus. The later work, node2vec [3] extends DeepWalk [9] by introducing hyperparameters that control a random walks tendency to conduct a Depth First Search (DFS) or a Breadth First Search (BFS) through the graph. They find that BFS helps to encode structural roles in a graph while DFS is most helpful for learning community structure.

While [3, 9] are concerned with learning only node representations sub2vec [1] can learn representations of subgraphs. If DeepWalk is to graphs as word2vec is to text then sub2vec is to graphs as doc2vec is to text documents. Just as the paragraph vector in [6] holds the "memory" of a document the subgraph vector in sub2vec maintains the "memory" of a subgraph. These two methods now seek to maximize the updated version of equation (1) with the subgraph vector s:

$$\arg\max_{\theta} \prod_{v_c, v_t}^{W} p(v_t | v_c, \mathbf{s}; \theta) \tag{2}$$

Other: To the best of our knowledge there is only one other paper that directly addresses the issue of cross-domain representations in the space of graph neural networks (GNNs). [8] introduces Deep Multi-Graph Embedding (DMGE) which relies on a multiple-gradient descent optimizer to balance learning across domains.

Not only do we learn cross-domain representations we also learn a representation for subgraph. Sub2Vec as detailed in [**?**] builds on the formalization in [9] which itself extends the skip-gram model [7] to arbitrary graphs.

3 THE PYPI GRAPH DATASET

The Python Package Index (PyPI) is home to over two-hundred and twenty-thousand python packages [1]. Finding similar packages in terms of use-cases, features, and performance can be very difficult without specific domain knowledge of a task or the python ecosystem. The PyPI Graph Dataset we provide language, and dependency graph domains gives us supergraph of python packages. With our goal being to jointly learn node and subgraph representations across multiple graph domains we have completely reinterpreted the GNN paradigm. As a result no previous dataset gives us all multiple raw graph domains to learn representations across, so it is necessary we introduce our own. While no dataset available gives us the information we need the class of problems that the PyPI Dataset represents is applicable to many other areas. For example, a research paper can be described by its location within a citation graph, an authorship graph, and by the language of the paper itself. A web page can be described by a hyperlink graph and the language on the web page. Representing these types of entities with multiple graphs, i.e. multiple domains requires learning representations that can cross domains and the PyPI Dataset gives a way to test methods that do just that.

3.1 The Language Graph

The language graph consists of either the package's ReadMe file or its PyPI description (whichever is longer) for 190k python packages. If we were to learn a python package representation from just this graph we would end up with a reasonable measure of package similarity. However, we reiterate that this one domain is a part to the whole concept that is a python package.

3.2 The Direct Dependency Graph

The language graph consists of either the package's ReadMe file or its PyPI description (whichever is longer) for 190k python packages. If we were to learn a python package representation from just this graph we would end up with a reasonable measure of package similarity. However, we reiterate that this one domain is a part to the whole concept that is a python package.

[1]See https://pypi.org/ for the most up-to-date statistics.

Table 1: Comparing PyPi Graph Dataset to other Graph Datasets with Text Features

Dataset	# Nodes	# Edges	# Text Feature Size	Features Format
Cora	2,708	5,429	1,433	Binary Keyword
Citeseer	3,312	4,732	3,703	Binary Keyword
PubMed	19,717	44,338	500	TF-IDF
PyPi Dependency Graph	**95,951**	**1,509,378**	**n/a**	**Raw Language**
Reddit	231,443	11,606,919	300	GloVe Embeddings

3.3 Data Collection

Data was collected using GitHub's GraphQl API and PyPI's API. We used PyPI as the entry point to every python package [2]. From there we can retrieve some textual data and links to GitHub repositories. Using GitHub's GraphQl API we have easy access to the direct dependencies found in files like requirements.txt, Pipfile, setup.py, etc. The ReadMe files on GitHub tend be much more detailed than the description on PyPi so we pull that along with topics set by the repository owner for potential downstream tasks.

3.4 Future Data to Collect

While we use just the language and direct dependency graph data we plan on adding more data domains in subsequent versions. The most obvious being the the code of the python packages. We do not yet have a quantitative measure of performance in this domain so we omit it from experiments in this paper. Additionally, another graph domain is a contributor graph where an edge exists between packages if they share a contributor. Lastly, we plan on adding non-graph data such as stars, PyPi downloads, number of versions, number of forks, etc. The abundance of potential data around python packages show that this is a worthwhile endeavor that we hope leads to more sophisticated methods that follow our supergraph interpretation.

4 CROSSWALK

In this section we introduce CrossWalk, an unsupervised model to learn representations across multiple graph domains. This fulfills our goal of being able to learn representations on graph data with graphical features. In the past, methods have relied on embedding the graphical features before any learning takes place ignoring any potential advantages of a jointly learned representation. In CrossWalk we use the raw data to learn these joint representations. Ultimately, there are two clear contributions of CrossWalk: (1) a new supergraph interpretation of graph data with graphical features and (2) a new random walk model that can learn representations across these multiple graph domains.

4.1 A New Interpretation

In the current GN paradigm a graph G has a set of vertices or nodes V, a set of edges E, and each node $v_i \in V$ has a feature matrix $\mathbf{X_i}$. CrossWalk deals with the case when v_i has graphical properties. Instead of first embedding these graphical properties into a vector that can fit in $\mathbf{X_i}$, CrossWalk aims to jointly learn a representation of the graphical properties of v_i and of G. To do this, we require a new interpretation.

Let G_s be a supergraph with the set of vertices V_s being the set of entities we would like to learn a representation for. Instead of a feature matrix $\mathbf{X_i}$ for each node in V, V_s is now connected to a set of graphs: $D = \{G_0...G_n\}$. Each $G_i \in D$ can be of two kinds - global or local. A global graph G_g has a set of edges E_g between the global supergraph nodes V_s (the entities we seek to represent) so there is no need for a new set of nodes. However, a local graph G_l has a new set of nodes V_l and - to relate the supergraph's nodes to this local graph - E_l also includes edges between V_l and V_s in addition to edges between V_l.

This interpretation successfully replaces $\mathbf{X_i}$ with the raw graph data that would've been represented separately. We can now focus on learning within this new paradigm.

4.2 The Random Walk Model

With this new interpretation a simple and natural evolution of past random walk methods emerge.

It has been empirically showed that the skip-gram model [7] can be applied to learn node representations from the social structure inherent to any arbitrary graph [9]. By taking random walks through the graph, beginning at a vertex of interest, DeepWalk maximizes the likelihood of observing a context vertex v_c conditioned on a target vertex v_t w.r.t the parameters θ. The objective can then be characterized by the following:

Where W is the set of all context target vertex pairs from random walks. However, this objective cannot learn a subgraph representation. Sub2Vec [?] generalizes doc2vec with the following objective:

Where s is a vector that characterizes a specific subgraph. Intuitively, this is the same objective that governs the Doc2Vec method simply generalized to arbitrary graphs.

Our maximum likelihood w.r.t the parameters of our model across all domains present becomes:

$$\arg\max_{\theta} \prod_{i}^{D} \prod_{v_c, v_t}^{W^{(i)}} p(v_t | v_c, \mathbf{d}; \theta) \tag{3}$$

Where D is the set of all domains, $v_t^{(i)}$ is the tth vertex in the ith domain, and $\mathbf{d}$ is the document vector.

Equation 3. can be further extended to include domain-specific embeddings and cross-domain embeddings. For example, with the PyPI dataset we could learn an embedding that crosses the language and dependency graph domains but remains independent of the code graph. In fact, the total possible number of embeddings that

[2] See: https://pypi.org/simple

Table 2: Topic Classification of Nodes

Domain	Text Only	Graph Only	Joint
Micro-F1:	44.96%	34.74%	46.89%
Macro-F1:	34.61%	21.20%	36.22%

can be learned with CrossWalk over all domains D is:

$$N_e = \sum_{k=1}^{D} \binom{D}{k} = 2^{D-1} \tag{4}$$

However, we highly encourage practitioners to not waste resources on learning all possible embeddings but to rather use this property of CrossWalk to determine which intersection of domains is best for a downstream task.

5 REPRESENTATION EVALUATION

5.1 Topic Classification

In order to evaluate the learned python package embeddings we use an extrinsic evaluation task. We define this task as the classification of python packages into different categories. GitHub allows for users to label their projects under various tags - as a vast majority of PyPI packages link to a GitHub repository we were able to extract a subset of python packages that are labeled by the creator into these topics. GitHub encourages users to classify their repositories "to help other people find and contribute to your project, you can add topics to your repository related to your project's intended purpose, subject area, affinity groups, or other important qualities." [3] This further justifies our interpretation of topics as classification classes.

Following the example of [9] (maybe add others?) we use a one vs. rest logistic regression model and a shallow neural network to classify the learned representations. These model choices reflect our desire to intentionally cripple the transformation ability of the classifier so that the natural discriminatory ability of the learned embeddings is what is being evaluated.

This task empirically shows that no single domain is sufficient to capture a complete representation of a python package. [Perform Ablation Study!]

5.2 Link Prediction

Topic Classification has a close association with the language domain in the PyPI graph dataset. To have a more complete benchmark of the dataset we use a link prediction task within the dependency graph domain. This tasks consists of predicting whether or not a connection should exist between node v and node u with inputs being the concatenation of the respective node embeddings.

To learn the mapping between embeddings and the existence of an edge we use a shallow NN implemented through sklearn's MLP classifer. Using 100k randomly selected edges we create another 100k false edges by randomly selecting source and destination nodes from the true edges. The values reported are the macro and micro F1 scores. We compare single domain representations created from only textual data and only graphical data with a jointly learned CrossWalk embedding.

[3]https://help.github.com/en/github/administering-a-repository/classifying-your-repository-with-topics

Table 3: Link prediction results on PyPi Graph Dataset (Dependency Graph Domain)

Domain	Text Only	Graph Only	Joint
Micro-F1:	65.94%	88.71%	86.52%
Macro-F1:	65.93%	88.70%	86.52%

The results in Table 3. show that while the text-only embeddings do 16% better than chance the graph embedding does the best which makes sense as this task comes from the dependency graph domain of the PyPI dataset. Most notably, the jointly learned embedding performs at just about the same score as the graph only domain. This joint representation is the same one used in the previous topic prediction task where the graph embedding did not do nearly as well as the text embedding. This shows we have achieved our goal of learning a single representation from raw graph data that captures information across multiple domains.

6 CONCLUSION

ACKNOWLEDGMENTS

REFERENCES

[1] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. 2017. Distributed representations of subgraphs. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 111–117.

[2] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2Vec: Learning Distributed Representations of Code. *Proc. ACM Program. Lang.* 3, POPL, Article 40 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290353

[3] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[5] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:cs.LG/1609.02907

[6] Quoc V. Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. *CoRR* abs/1405.4053 (2014). arXiv:1405.4053 http://arxiv.org/abs/1405.4053

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[8] Yi Ouyang, Bin Guo, Xing Tang, Xiuqiang He, Jian Xiong, and Zhiwen Yu. 2019. Learning Cross-Domain Representation with Multi-Graph Neural Network. *arXiv preprint arXiv:1905.10095* (2019).

[9] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD '14)*. ACM, New York, NY, USA, 701–710. https://doi.org/10.1145/2623330.2623732

[10] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations* (2018). https://openreview.net/forum?id=rJXMpikCZ accepted as poster.

[11] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu

A REPRODUCIBILITY

There is a much needed and growing movement in the field of machine learning that encourages the researchers to ensure all results are easily reproduced. In the context of this paper and our results we make this priority.

We want to help create a standard that moves beyond providing a script or notebook that can be run that output the same numbers found in the paper. Instead, we want to build a tool that will allow other researchers to not just match our numbers but build on any aspect of our codebase that may benefit future research.

To accomplish this goal we highlight the following features in our codebase that will be open to the public upon the publication of this paper. (1) We provide an API that makes it easy for users to switch out datasets beyond those used in this paper. (2) We provide tooling to explore the embedding spaces learned via CrossWalk in the form of a web API. (3) We prioritize the use open source libraries and frameworks so no software used is hidden behind a proprietary wall.