CrossWalk: Learning Representations Across Multiple Graphs

Anonymous

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 e.g. a python package, a user in social media network, or a webpage. In this paper, we help address this problem with a two-fold contribution: (1) We introduce the PyPI graph dataset that includes multiple graph data domains. (2) We benchmark this dataset with a novel unsupervised method that extends random walk models to cross multiple domains which we coin CrossWalk.

CCS CONCEPTS

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

ACM Reference Format:

Anonymous. 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, 6 pages. https://doi.org/10.1145/1122445.1122456

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 graph representation learning methods [5, 7, 15] interpret a graph G as a set of vertices or nodes V, a set of edges E, and a feature matrix $\mathbf{X_i}$ for each node $v_i \in V$. In the case that each node has graphical features, one must first learn a vector representation to place in each $\mathbf{X_i}$. For example, in a citation graph each node has textual data, i.e. the paper itself, which can be interpreted as a language graph [12], but in the $\mathbf{X_i}$ paradigm this is not acknowledged. Instead, popular datasets used in graph representation learning literature provide premade embeddings in place of learning another graph representation. As a result, methods that follow the $\mathbf{X_i}$ paradigm cannot learn from the raw data of multiple graphs directly.

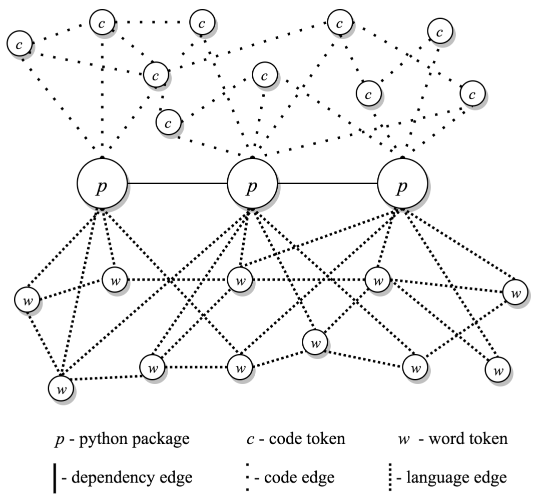

Figure 1: CrossWalk's interpretation of the PyPI graph dataset. Notice the three distinct graphs, i.e domains, and how each relate to a python package.

Furthermore, these datasets poorly represent a class of problems where multiple graphs describe a single entity. A python package is such an entity as each package is characterized by a language graph from its documentation, a code graph, a dependency graph, and more. In this example, no single graph can claim to be sufficient to fully represent a python package. Therefore, we require a model that can learn representations which encode the relevant information across multiple graph domains directly from raw data.

In response to these issues, we first present a new interpretation that allows for learning across multiple graph domains from raw data. Discarding the $\mathbf{X_i}$ interpretation, let G_s be a supergraph with a set of vertices V_s being the set of entities we would like represent. Instead of a requiring a feature matrix, $\mathbf{X_i}$, for each node, 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 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 within V_l.

In the case of our python package example, each node in V_s corresponds to a python package in our dataset and D contains three graphs: a language graph, a code graph, and a dependency graph (See Figure 1). 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 word nodes the python package representation must act as a document or, more generally, a subgraph vector as described in doc2vec[8] or sub2vec[1]. The same is done for the code graph only each local node is a syntax token (perhaps taken from an abstract syntax tree [2]). With this problem formulation a good python package representation must encode the relevant information in all three graphs.

This new interpretation is the first contribution of the CrossWalk model. The second is the extension of random walk methods [1, 3, 8, 9, 12] to learn these cross-graph representations from raw graph data. Exactly how this is done is left to Section 4.

Lastly, to validate the CrossWalk model, we require a dataset with all raw information available. Unfortunately, the datasets consistently referenced in graph representation learning literature have only one graph domain or provide document embeddings when language graphs are present. In response, we have created our own dataset of python packages - the PyPI graph dataset. The PyPI dataset currently has ~200k python packages with two graph domains: (1) a language graph and (2) 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 raw graph data with raw graphical features.

2 RELATED WORK

In recent years, graph representation learning has narrowed in on random walk methods and graph convolutional networks (GCNs). Random walk methods build a corpus of fixed-length random walks through a graph and usually do not rely on node features. Meanwhile, GCNs aggregate local node features generalizing the idea of convolutional neural networks (CNNs) commonly used in the field of Computer Vision. As the model in out dataset extends random walk methods we discuss past work in this area below. For a complete review of graph representation learning we refer the reader to the following articles [6, 16, 18].

DeepWalk [12], node2vec [3], and sub2vec [1] all extend Mikolov et al.'s skip-gram language model [9] to arbitrary graphs with the use of random walks. A random walk begins at a randomly chosen node v, then steps to another randomly chosen node from v's first-degree neighbors and so forth, until a given length is met. As pointed out in [12], a sentence is analogous to a random walk through a language graph thus, allowing for the application of the skip-gram language model [9] to a corpus of random walks to learn node representations. As a result, 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 contains all context and target node pairs, (v_c, v_t), over all windows in a random walk corpus. The later work, node2vec [3] extends DeepWalk [12] by introducing hyperparameters that control a random walks bias to walk via a Depth First Search (DFS) or a Breadth First Search (BFS) through the graph. They find that BFS better encodes structural roles in a graph while DFS is better for learning community structure.

While [3, 12] learn node representations, sub2vec [1] learns subgraph representations as the name suggests. If we relate these methods to language models, DeepWalk is to graphs as word2vec is to text and sub2vec is to graphs as doc2vec is to documents. Just as the document vector described in [8] holds the "memory" of a document the subgraph vector in sub2vec holds the "memory" of a subgraph. These two methods now seek to maximize the updated version of equation 91) with a subgraph vector $\mathbf{s}$ that represents the subgraph from which v_c and v_t belong:

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

This paper extends random walk methods discussed above to learn across multiple graphs. Learning over multiple graphs seems to be a sparse area of research, but we identify two articles in this area. First, Deep Multi-Graph Embedding (DMGE) [11] introduces a multiple-gradient descent optimizer to balance learning across domains. However, this method assumes only one global set of nodes and therefore, cannot directly incorporate subgraph representations into their joint embeddings (which is essential for learning from a language graph). Second, LinkNBed [14] uses multiple knowledge graphs to learn relational score functions between entities. While very interesting, this method is so closely tied to knowledge graphs that it almost falls outside the scope of this paper. To the best of our knowledge CrossWalk is the first model to learn across multiple global and local graph domains via random walk methods.

3 THE PYPI GRAPH DATASET

The Python Package Index (PyPI) is currently home to over two-hundred and twenty-thousand python packages [1]. With this new dataset, the field of machine learning can now benefit from the diverse and unique data found in the python package ecosystem, especially, when applying machine learning to the abundant graphical data. Furthermore, the PyPI graph dataset is representative of a class of entities where just one graph domain is not sufficient. For example, a research paper can be described by its location within a citation graph, an authorship graph, and by a language graph of the paper itself. Or, a webpage, which can be described by a hyperlink graph and a language graph. Representing these types of entities with multiple graphs requires learning representations that can cross graph domains. This is the key difference between popular datasets like Cora, Citeseer, PubMed, etc., and ours (See Table 1). Now, researchers can apply models that learn a joint representation, combining the two domains in this dataset, instead of accepting premade embeddings. In the following we discuss the two graph domains in this initial version of the PyPI graph dataset:

3.1 The Language Graph

In this domain, we chose to interpret language as a graph just as DeepWalk has. This language graph consists of 190k ReadMe files with a large and diverse vocabulary. Notably, as these files act as the first step to understanding what a python package is they are a valuable source of data when learning python package representations. If a python package representation was learned from just this graph we would end up with a reasonable measure of package similarity. However, we reiterate that this domain is just one part of the whole python package entity.

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

Table 1: Comparing the PyPI graph dataset to popular graph datasets

Dataset	# Nodes	# Edges	#Features	Text 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	602	GloVe Embeddings

3.2 The Direct Dependency Graph

What makes python such an incredibly popular language is its ease use and massive ecosystem of packages. When one package, v_0, imports another package, v_1, then v_0 is said to depend on v_1, and as a result, a massive dependency graph is formed over all packages. To make inferences about each node in this graph, models can build on the intuition that similar packages probably depend on similar packages. In the PyPI graph dataset, the direct dependency graph is stored as a list of edges between python packages and contains ~95k nodes with ~1.5 million edges between them.

3.3 Data Collection

All data was collected using PyPI's API and GitHub's GraphQl API. We used PyPI as the entry point to every python package [2]. From there, textual data and hyperlinks to GitHub repositories was retrieved. Then, these hyperlinks and GitHub's GraphQl API were used to gather direct dependency data which is found in files like requirements.txt, Pipfile, setup.py, etc. Additionally, ReadMe files were gathered from these GitHub repositories. Lastly, we collected topics set by repository owners to act as labels in downstream tasks like node classification.

3.4 Future Data to Collect

While this initial version contains 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. Additionally, an anonymized contributor graph domain where an edge exists between packages if they share a contributor would be a beneficial addition to this dataset. Lastly, we plan on adding non-graph data such as number of GitHub stars and forks, PyPI download statistics, package versioning statistics, etc. The abundance of potential data around python packages show that this is a worthwhile endeavor that we hope results in more sophisticated graphical machine learning methods.

4 CROSSWALK

In this section we introduce CrossWalk, an unsupervised model to learn representations across multiple graph domains. In the past, graph representation learning 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, even with language. Ultimately, there are two clear contributions of CrossWalk: (1) a supergraph interpretation of graph data with graphical features and (2) a random walk model that can learn representations across these multiple graph domains.

[2]See: https://pypi.org/simple and https://developer.github.com/v4/

4.1 A New Interpretation

In the current graph representation learning 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 or is a part of multiple graphs. Instead of first embedding these graphical properties into a vector that can fit in $\mathbf{X_i}$, CrossWalk aims to jointly learn these representations. 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 represent. Instead of a feature matrix $\mathbf{X_i}$ for each node in V, V_s is now connected to a set of graphs, i.e. domains: $D = \{G_0...G_n\}$. 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 supergraph nodes, V_s (the entities to represent), therefore, 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 within V_l. Intuitively, we learn a node representation within global graphs and a subgraph representation of local graphs. Because of this, CrossWalk is a direct generalization of both DeepWalk and sub2vec.

This new interpretation successfully replaces $\mathbf{X_i}$ with the raw graph data that would've needed an independent and prior representation from the final one. We now focus on learning within this new paradigm.

4.2 Our Random Walk Model

From this new interpretation, a simple and natural evolution of past random walk methods emerge. We extend the likelihood equation (2) across all of our graph domains D. Where $W^{(i)}$ is the set of all context, target vertex pairs in the ith domain created via windows over random walks (See Appendix for all hyperparameter details). Therefore, our maximum likelihood loss across all graph domains becomes:

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

In our formulation, we choose to optimize θ through negative sampling, a form of noise contrastive estimation [4, 9, 10]. It then

Table 2: Task and Average Task Performance on PyPI Graph Dataset

Domain:	Text	Dependency Graph	Joint	$\mathbf{X_i}$ Joint
Method:	Doc2Vec	DeepWalk	**CrossWalk**	GraphSAGE
Node Classification Performance				
Micro-F1:	44.96%	34.74%	**46.89%**	17.80%
Macro-F1:	34.61%	21.20%	**36.22%**	3.83%
Link Prediction Performance				
Micro-F1:	65.94%	**88.71%**	86.52%	76.56%
Macro-F1:	65.93%	**88.70%**	86.52%	75.50%
Average Performance				
Micro-F1:	55.45%	61.73%	**66.71%**	47.18%
Macro-F1:	50.27%	54.95%	**61.22%**	39.67%

follows that our loss function across multiple graph domains becomes:

$$\sum_{i=1}^{D}[\log\sigma\left(v_c^{(i)}\cdot v_t^{(i)}\right)+\sum^{k}\log\sigma\left(v_{neg}^{(i)}\cdot v_t^{(i)}\right)] \tag{4}$$

Where k is the number of negative samples and $v_{neg}^{(i)}$ is a negatively sampled vertex vector from domain i. As discussed in Section 4.1, a domain can be local or global, therefore θ consists of a single set of global parameters and a set of local parameters for each local domain. As a result, the number of parameters in CrossWalk scales linearly w.r.t to the number of node in each local domain, but stays constant w.r.t to the global domains. Alternatively, one could chose to introduce parameters that represent a subset of graph domains in D. For example, if $D=\{G_0,G_1,G_2\}$ one could instantiate a set of parameters, $\theta_{0\cup 2}$, that will only cross G_0 and G_2, ignoring data from G_1. As a result of this property, CrossWalk can learn up to 2^{D-1} sets of parameters, i.e. representations:

$$N_\theta=\sum_{k=1}^{D}\binom{D}{k}=2^{D-1} \tag{5}$$

However, even though one can doesn't mean one should. Clearly, this growth rate w.r.t D is massively expensive in terms of memory and gradient updates. Instead, the benefit of this CrossWalk property is that researchers can use task-specific knowledge to learn multiple representations over various subsets of D. These could then be used in downstream tasks to evaluate the necessity of certain graph domains. For example, if $\theta_{0\cup 2}$ does just as well on some task as θ_g then there is evidence G_1 is not necessary for such a task.

4.3 Future Work

Currently, CrossWalk omits the case when there is non-graphical data present. These features like number of GitHub stars or number of GitHub forks will require an updated version of CrossWalk. We leave this to future work alongside the collection of such data for the PyPI graph dataset. We also acknowledge that it could be beneficial to some tasks to allow for supervised learning. Naively, this could be achieved by adding a loss function for a task to the CrossWalk negative sampling loss (equation 4). Lastly, when learning across multiple domains some domains may not provide a balanced amount of data which means certain domains could be over represented in the final embedding vector. Future work could include applying multi-task learning solutions[13] to this potential problem.

5 REPRESENTATION EVALUATION

In order to evaluate the learned python package representations we use two extrinsic evaluation tasks in each domain of the PyPI graph dataset: (1) Node classification that favors the textual domain and (2) link prediction in the dependency graph domain. We can report that only CrossWalk learns python package representations that achieve high scores across both tasks (Table 2).

5.1 Node Classification

In the context of the PyPI graph dataset, we define this task as the classification of python packages into different topics. The topics were collected from the GitHub repositories and act as labels for ~20k python packages in our dataset. GitHub encourages users to classify their repositories stating, "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] Which provides justification for our use of topics as node labels.

Following the example of [12], we use a one vs. rest logistic regression model to classify the learned representations and report the macro and micro F1 scores in Table 2. This model choice reflects our desire to intentionally cripple the transformation ability within the classifier so that the natural discriminatory ability of the learned embeddings is evaluated.

We compare CrossWalk to: DeepWalk, Doc2Vec, and unsupervised GraphSAGE[5]. DeepWalk and Doc2vec can only represent the graph and language domain respectively. Note that CrossWalk learned over a single graph domain is equivalent to DeepWalk and

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

equivalent to Doc2Vec when learned over a language graph. By using these single-domain variants of CrossWalk, we show that CrossWalk's joint embedding matches single-domain performance while still representing multiple domains. In this task, the joint embedding beats out Doc2Vec which provides evidence that the dependency graph domain contributes valuable information - this would not be possible without learning across multiple domains. We also compare CrossWalk to an unsupervised GraphSAGE implementation where each node is associated with a TF-IDF vector of length 256 to capture relevant language information in its application to the dependency graph. The poor performance of GraphSAGE in this task validates our assertion that the $\mathbf{X_i}$ interpretation does not match the quality of representations that directly incorporate multiple graph domains.

5.2 Link Prediction

To have a more complete benchmark of the PyPI graph 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 two node from 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 shuffling source and destination nodes of the true edges. The values reported are the macro and micro F1 scores. We compare methods using the same representations as in the node classification task because a quality python package representation must encode information relevant to multiple graph domains as discussed in Section 3.

The results in Table 2. show that while CrossWalk's joint representation performs slightly worse than the graph-only representation it is still comparable. This difference in performance might be attributed to the joint representation having to balance the textual domain as well which indicates that the textual domain is not beneficial for this task. Nevertheless, the performance of the joint embedding still outperforms the $\mathbf{X_i}$ interpretation of GraphSAGE - lending additional validation to CrossWalk's novel interpretation.

5.3 The Key Point

We end our representation evaluation section by reiterating that only CrossWalk's joint representation was able to perform among the best in both of these tasks as shown by the average performance in Table 2. A python package representation that cannot perform well in both of these tasks does not adequately represent the entity at hand. From this, we can conclude that these results validate CrossWalk's novel interpretation and random walk method.

6 CONCLUSION

CrossWalk introduces a new interpretation of graphs with graphical features that allow us to learn joint representations across graph domains from raw data. Due to this new interpretation current datasets used in the graph representation learning literature do not suffice. As a result, we introduce the PyPI graph dataset with raw text and graph data (with more to follow) that is representative of a class of problems where one graph data domain is not sufficient to represent the whole, i.e. research papers, webpages, social media users, etc. CrossWalk also provides a simple extension of past random walk models so that we can learn under this new interpretation. We then apply CrossWalk to the PyPI dataset and find that CrossWalk successfully learns representations that encode the information of the multiple graph domains it was trained on.

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] Michael U Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research* 13, Feb (2012), 307–361.

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

[6] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

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

[8] 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

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[10] Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426* (2012).

[11] 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).

[12] 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

[13] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[14] Rakshit Trivedi, Bunyamin Sisman, Jun Ma, Christos Faloutsos, Hongyuan Zha, and Xin Luna Dong. 2018. Linknbed: Multi-graph representation learning with entity linkage. *arXiv preprint arXiv:1807.08447* (2018).

[15] 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.

[16] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596* (2019).

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

[18] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

A REPRODUCIBILITY

There is a much needed and growing movement in the field of machine learning that encourages 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 simply outputs the same numbers found in the paper. Instead, we have built 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.

Table 3: Hyperparameters used

Domains:	Text Only	Graph Only	Joint
Epochs:	20	20	15
# Examples:	26,876,633	34,529,040	61,405,673
# Negative Samples:	2	2	2
Optimizer:	Adam	Adam	Adam
Learning Rate:	1e-3	1e-3	1e-3
Batch Size:	4096	4096	4096
Window Size:	5	5	5
Embedding Size:	128	128	128

To accomplish this goal we first note that all of our code will be published on GitHub after the anonymous review process has been completed. Second, we will do the same for the PyPI graph dataset we introduce.

A.1 The Code

The first section of our codebase includes hyperparameters and all decisions made that may affect any of the results shown. This includes jupyter notebooks that run the two evaluation tasks on the PyPI dataset so curious readers can see the results reported. The second section acts as an easily extendable model so users can learn on their own datasets or build better methods without having to start from scratch.

All code is written in python and training is done using the PyTorch framework (except in some evaluation scripts). We provide an explicitly versioned python environment as a Pipfile to ensure experiments aren't confounded by versioning issues. We also include all data prepossessing scripts that explicitly detail random walk and tokenization procedures. Additionally, we provide the ability to train from HDF5 files if all training examples are too large to fit into RAM. Our choice to provide such features reflects our goal to allow users with varying levels of compute power be able to replicate our results and use their own data with ease. The results reported in this paper were all conducted on a 1080ti GPU with 32Gb or RAM.

A.2 The Dataset

In addition to releasing the raw data, we also plan on releasing the python scripts used to collect the data. As we are planning subsequent versions of the PyPI dataset, we believe it is important to release these scripts to account for any differences between dataset versions. We also believe that allowing users to improve these scripts through an open source project will result in a stronger dataset and stronger models.

A.3 Experiment Hyperparameters

For the two experiments detailed in this paper we built one set of representations: text only, dependency graph only, and joint. We used the hyperparameters reported in Table 3. The random walk and tokenization procedure used to create training examples in the graph and text domains respectively is detailed in the codebase mentioned above.

A.4 Node Classification Task

In the node classification task, we report the average of 10 random train/test splits. The model used was implemented with sklearns logistic regression API and optimized with its lbgfs solver. We adapt DeepWalk's codebase[4] for scoring this task.

A.5 Link Prediction

In the link prediction task we report the average of 10 results on 100k separate randomly selected edges. The shallow neural network model used was implemented with sklearns MLP API, had 100 hidden units, and optimized with its lbgfs solver.

[4]https://github.com/phanein/deepwalk/