Learning Distributed Representations of Graphs with Geo2DR

Paul Scherer
University of Cambridge
Cambridge, United Kingdom

Pietro Liò
University of Cambridge
Cambridge, United Kingdom

ABSTRACT

We present Geo2DR (*Geo*metric to *D*istributed *R*epresentations), a GPU ready Python library for unsupervised learning on graph-structured data using discrete substructure patterns and neural language models. It contains efficient implementations of popular graph decomposition algorithms and neural language models in PyTorch which can be combined to learn representations of graphs using the distributive hypothesis. Furthermore, Geo2DR comes with general data processing and loading methods to bring substantial speed-up in the training of the neural language models. Through this we provide a modular set of tools and building blocks to quickly construct methods capable of learning distributed representations of graphs. This is useful for replication of existing methods, modification, and development of completely new methods. This paper serves to present the Geo2DR library and perform a comprehensive comparative analysis of existing methods re-implemented using Geo2DR across widely used graph classification benchmarks. Geo2DR displays a high reproducibility of results of published methods and interoperability with other libraries useful for distributive language modelling, making it a useful addition to the graph representation learning toolkit.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; *Kernel methods*; **Neural networks**; **Learning latent representations**.

KEYWORDS

library, toolkit, graph representation learning, reproducibility, distributed representations

ACM Reference Format:
Paul Scherer and Pietro Liò. 2020. Learning Distributed Representations of Graphs with Geo2DR. In *DLG-KDD'20: The Second International Workshop on Deep Learning on Graphs: Methods and Applications, August 24, 2020, San Diego, CA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Representation learning of graphs using neural networks has turned into a large and exciting hub of research driven by successive proposals of graph representation learning methods and datasets to apply them onto. A significant part of the activity has focused on *Graph Convolutional Neural Networks* (GCNN). Such neural networks are characterised by *graph convolutional* operators [2, 6, 13] that serve as useful inductive biases for learning representations of nodes and other graph substructures. Gilmer et al. [8] generalised the convolution operator over irregular domains as a message passing scheme (MPNN), allowing the specification of a full spectrum of methods as variants of this equation. Representations of entire graphs are then created through the successive application of graph convolution operations followed by different *pooling* methods [6, 15, 28] which aggregate node representations towards a single vector representation for the entire graph.

The difficulty of reliably constructing GCNN models has driven the need for toolkits and libraries to facilitate their development for replication, extension and creation of new models. Several such libraries have been made such as: *Graph Nets* introduced by Battaglia et al. [1], *DGL* by Wang et al. [26], *GEM* by Goyal et al. [9], and most recently *PyTorch Geometric* by Fey and Lenssen [7]. These libraries have greatly contributed to lowering the barrier of entry into GCNN research, fueling the development of novel methods and libraries supporting them in a healthy feedback cycle.

Alongside ongoing research into GCNNs and its variants, another approach has focused on extending graph kernel methods with neural language embedding methods [11, 17, 27] that exploit the distributive hypothesis to learn representations of graphs. This is a useful alternative inductive bias to model the vector space embeddings of graphs over the distribution of the discrete substructure patterns *contextualising* them. Much like how the semantic meaning of words is similar to words that have similar context words around them [10], comparability can also be defined for graphs with the appropriate specification of what constitutes context and the entities (nodes, subgraphs, substructure patterns) that are involved. Such vector representations of graphs are inductively biased to be close when they contain similar substructure patterns, and distant when they do not. This perspective enables the construction of a powerful class of unsupervised representation learning methods.

At this point, there are multiple excellent MPNN and Graph Kernel libraries for calling specific implementations of existing methods. Some GCNN focused libraries such as PyTorch Geometric [7] even allow composing new methods by interfacing with extensible message-passing or pooling modules. However, to our knowledge, no toolkit currently exists for rapidly composing *new* methods capable of learning distributed representations of graphs. This project, Geo2DR (*Geo*metric to *D*istributed *R*epresentations), aims to fill this gap by providing a modular set of building blocks built around a conceptual framework that is applicable to existing methods and an even greater number of unexplored ones. The Geo2DR library along with links to documentation, example methods reimplementations, experiment replication, and supporting material can be found on the GitHub repository (**https://github.com/paulmorio/geo2dr**) with package releases on PyPI.

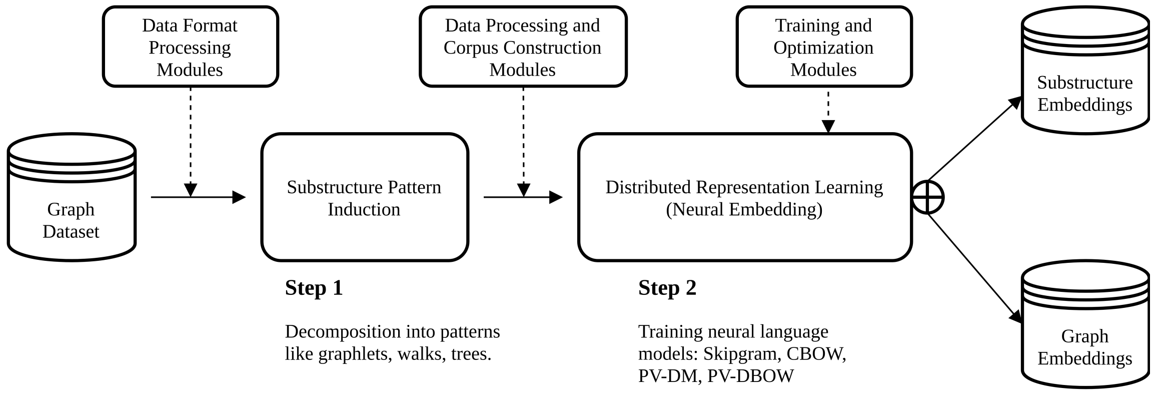
Data Format Processing Modules
Data Processing and Corpus Construction Modules
Training and Optimization Modules
Substructure Embeddings
Graph Dataset
Substructure Pattern Induction
Distributed Representation Learning (Neural Embedding)
Step 1
Decomposition into patterns like graphlets, walks, trees.
Step 2
Training neural language models: Skipgram, CBOW, PV-DM, PV-DBOW
Graph Embeddings

Figure 1: The two-stage design methodology for creating distributed representations of graphs and the various modules (in rectangles) included in Geo2DR to support this process. All modules were designed with consistent interfaces so that they may be mixed and matched to create existing and novel methods, as well as simplify integration of custom modules.

2 BACKGROUND

The approach towards distributive modelling of graphs was pioneered by Yanardag and Vishwanathan [27]. They observed that many graph kernel methods can be formulated as instances of the R-Convolutional framework. Herein, the similarity between different graphs is computed by decomposing graphs into discrete substructure patterns such as graphlets, shortest paths, and rooted subgraphs. This produces a $|\mathbf{V}|$-dimensional bag-of-words or pattern frequency vectors for each graph where $\mathbf{V}$ is the set of the unique patterns induced over all the graphs in a dataset. The graphs and their induced substructure patterns are input to a kernel function, such as counting the common substructures across pattern frequency vectors. This defines the relation or similarity measure between the graphs to construct the kernel matrix for use with kernel methods such as SVMs.

Yanardag and Vishwanathan [27] further observed that as the size of graphs and the specificity of substructure patterns to be induced from graphs increases (via lengthening walks/paths, increasing the number of nodes in graphlet patterns) graphs are represented by extremely high dimensional pattern frequency vectors. As a result, only few substructure patterns are common across any given set of graphs producing sparse solutions where each graph is more similar to itself, a phenomenon known as *diagonal dominance.* To tackle this issue the authors proposed the use of neural language models which exploit the distributive hypothesis [10] to learn smooth low dimensional *distributed representations* of the substructures and construct graph kernel matrices. This was quickly followed up by works such as the aptly named Graph2Vec [17] and Anonymous Walk Embeddings [11] (AWE). These proposed different substructure patterns to induce over the graphs and the use of Doc2Vec variants [14] to build distributed representations of whole graphs directly. We provide a brief primer and conceptual framework for learning distributed representations of graphs in Appendix A.

Geo2DR provides various modules that can be used as "building blocks" to rapidly construct systems capable of learning such distributed representations of both substructure patterns and whole graphs of arbitrary size. Existing libraries for GCNNs [1, 7, 9, 26] would require a substantial shift in philosophical focus from constructing message passing schemes and pooling methods to accommodate these methods. Hence Geo2DR is a complementary library alongside existing toolkits enabling researchers a broader range of options and tools for graph representation learning. A brief comparison of existing libraries for graph representation learning is provided in Section 5 after describing the structure and usage of Geo2DR for better exposition.

3 OVERVIEW OF GEO2DR

Geo2DR is a Python library containing various "building blocks" to support rapid construction of methods capable of learning distributed representations of graphs. This framework for self supervised learning of substructures and entire graphs is based around a simple two stage design methodology summarised in Figure 1.

- **Induction of descriptive substructure patterns**: The first step consists of inducing discrete substructure patterns such as graphlets, rooted subgraphs, or anonymous walks within and across the dataset of graphs to construct a shared vocabulary and *corpus* dataset contextualizing the patterns and graphs. One may also use the output pattern distributions at this stage to construct a variety of graph kernels.
- **Learning distributed vector representations**: The second stage consists of utilising the distributive hypothesis [10] to learn distributed representations of graphs contextualised by the induced substructure patterns. Embedding methods which exploit the distributive hypothesis such as skipgram [16] can be used to learn fixed-size vector embeddings of substructure patterns or whole graph in an unsupervised manner.

The two stage methodology allows for the succinct description of existing methods as compositions of what substructure patterns are being induced across the graphs, and the specification of the target-context relationships as implied by the distributive neural embedding method. Hence, combination of Geo2DR's modules for

Table 1: Table characterising each of the existing published methods by the substructure patterns induced and associated embedding method to create the graph kernel matrix (for DGK models) or graph embeddings.

Method	Induced substructure pattern	Embedding method	Object embedded
DGK-WL	WL rooted subgraphs	Skipgram or CBOW	Substructure patterns
DGK-SP	Shortest paths	Skipgram or CBOW	Substructure patterns
DGK-GK	Graphlets	Skipgram or CBOW	Substructure patterns
Graph2Vec	WL rooted subgraphs	PV-DBOW	Whole graphs
AWE-DD	Anonymous walks	PV-DM	Whole graphs

decomposition and distributed representation learning can be used to quickly replicate existing methods such as those shown in Table 1. Just as importantly, it highlights the vast possibilities for the development of novel methods intersecting ongoing research in graph theory and distributive modelling through focused development of the modules.

Consistent input/output interfaces were implemented across modules to encourage creation of novel methods. For example, one could create a "novel" unpublished method combining existing modules on inducing shortest path patterns and learning graph-level embeddings with PV-DBOW. This form of light experimentation fosters understanding and control of the various inductive biases involved when building such models. However, in a more far-sighted view, we hope it would also encourage the creation of custom modules that can plug and play with the rest of the framework to create truly novel methods down the line.

Practically, the library is centered around three subpackages under Geo2DR. The `data` subpackage, contains modules for transforming data formats used by popular dataset repositories such as Kersting et al. [12] into consistent formats used by the decomposition algorithms implemented in Geo2DR. In Geo2DR, we chose to use the GEXF (Graph Exchange XML Format) as permanent storage format for individual instances of the graphs. This is because the format is compatible with network analysis software such as Gephi and NetworkX for detailed inspection.

The modules within the `decomposition` subpackage contain algorithms for inducing the substructure patterns in the graphs and forming vocabularies. The outputs of these algorithms are directly compatible with our PyTorch implementations of neural language models to utilize GPUs as well as those in Gensim [19]. This essentially describes the packages and modules necessary for Step 1 of the process. The final subpackage `embedding_methods` contains modules for constructing corpus datasets and neural language models to build the distributed representation learning methods of Step 2. Several `Trainer` classes are also included which serve as battery-included corpus and neural net combinations that can be used to construct common architecture setups.

Existing methods for learning distributed representations as in Table 1 and several graph kernels can all be implemented using the modules and frameworks presented. We have included all these methods as examples within the repository to get users started on creating their own variations. A brief code example using Geo2DR is provided in Appendix B.

4 EMPIRICAL EVALUATION

As a form of validation on the correctness for the various implemented modules, we empirically evaluate re-implementations of existing models using Geo2DR. Table 1 describes the induced substructure pattern and neural language model driving each method. We performed a series of common benchmark graph classification tasks under homogeneous data and evaluation scenarios giving a fairer picture of how they compare.

All datasets were downloaded from the benchmark dataset repository by Kersting et al. [12] and processed into the format used by Geo2DR with the included data formatter. In each of the datasets the discrete node labels are exposed, but not the edge labels. For unlabelled datasets such as REDDIT-B, the node was labelled by their degree following practice of Shervashidze et al. [21] to enable methods such as the WL rooted subgraph decomposition to induce patterns in the graphs; this was also applied to methods which can directly handle unlabelled graphs for conformity. As these datasets are standard benchmarks we have left specific descriptive details in Appendix C.

For all experiments, attempts were made to follow the hyperparameter setups described in the published papers of the original methods, with best-guess settings where details were unknown. As we look at several kernels and embedding models specific hyperparameter ranges can be found in Appendix D. In all cases, the same off-the-shelf support vector machine implemented in SciKit-Learn [18] was used with an RBF kernel trick for the supervised classification task on the graph embeddings learned. This SVM was chosen on the basis that all works used SVMs in their downstream classification tasks. C values were estimated over the set (0.001, 0.01, 0.1, 1, 10, 100). We report the average score of 10 iterations of training and applying 10 fold cross-validation using the SVM over random data splits with individual training restarts in all cases. The exact setups of the experiments can be replicated using the experiment replication code provided within the Github repository[1].

Graph kernels: We start with an experiment suite based on the substructure patterns alone, using the decomposition algorithms to construct normalised bag-of-words frequency vectors for each of the graphs. Table 2 records the mean and standard deviation of randomly split 10 fold cross-validation using the SVM described above. The results closely match that of the published methods in [3, 11, 21, 27]. The fact that different substructure patterns excel in classifying some datasets and do not perform as well in others

[1]https://github.com/paulmorio/geo2dr/tree/master/replication

Table 2: Random-split 10 fold cross-validation performance of SVM using RBF kernel on bag-of-words vectors of normalised frequencies of substructure patterns. Best scores or those within error of best are bolded. OOM denotes out-of-memory.

Substructure pattern	MUTAG	ENZYMES	PROTEINS	NCI1	REDDIT-B	IMDB-M
WL Rooted Subgraphs	**88.95 ± 7.96**	**56.33 ± 6.18**	74.29 ± 2.55	**83.94 ± 1.99**	77.35 ± 4.35	48.60 ± 4.33
Shortest Paths	83.68 ± 7.24	41.67 ± 4.83	**74.73 ± 2.04**	70.95 ± 1.95	OOM	**50.20 ± 3.84**
Graphlets	83.16 ± 6.16	25.33 ± 3.48	70.36 ± 3.59	54.09 ± 7.61	78.25 ± 2.71	44.40 ± 4.17
Anonymous Walks	80.53 ± 6.68	27.33 ± 6.23	71.87 ± 2.05	66.08 ± 2.21	**81.30 ± 2.49**	38.20 ± 3.91

Table 3: Graph classification performance over random-split 10 fold cross-validation in each of the re-implemented methods with standard deviation. Best scores or those within error of best are bolded. OOM denotes out-of-memory.

Method	MUTAG	ENZYMES	PROTEINS	NCI1	REDDIT-B	IMDB-M
DGK-WL	**88.42 ± 8.42**	41.00 ± 1.83	72.08 ± 0.74	**77.54 ± 3.91**	OOM	47.82 ± 0.79
DGK-SP	84.03 ± 7.16	44.27 ± 2.26	**76.93 ± 2.56**	69.22 ± 5.29	OOM	**49.71 ± 1.18**
DGK-GK	84.21 ± 6.74	23.61 ± 3.14	69.77 ± 3.13	53.92 ± 4.81	78.32 ± 1.92	44.40 ± 4.18
Graph2Vec	84.91 ± 2.79	**51.77 ± 1.75**	74.05 ± 2.28	71.34 ± 2.12	**81.25 ± 2.64**	47.11 ± 1.42
AWE-DD	79.29 ± 2.92	23.76 ± 1.74	69.70 ± 1.29	63.54 ± 1.82	**81.46 ± 1.75**	40.53 ± 6.42

suggests that topological characteristics which are useful for characterising graphs are not found in just one substructure pattern. Making the study of other patterns and combinations thereof an interesting avenue.

Deep graph kernels and graph embeddings: Most of our experiments in Table 3 show a high reproducibility of the results published by the original proposers. Some discrepancies are to be expected due to the homogenised data setup, unpublished hyperparameter settings, and standardised neural architectures, but best effort was made through consulting original source code and communications with the authors. In particular, for AWE-DD, we do not use edge-labels for homogeneity of the experiment evaluation whilst the original paper used them if they provided a better performance.

Runtime experiments and improvements in Geo2DR: Table 4 contains the average total training times incurred over 100 epochs, performed ten times with one standard deviation on a single quad-core Intel 4690 CPU. Comparison is drawn between the original reference implementation made available by each of the original papers and its re-implemented counterpart in Geo2DR. All methods were trained and compared on the MUTAG dataset as this was the only common dataset included in the reference implementations. None of the original reference implementations have scripts or tools to transform the publicly available datasets they used into the proprietary formats used by their own implementations, making reproduction difficult. This is why we have included data processing tools for popular public datasets directly into the Geo2DR library within the `data` subpackage to address this common limitation for the future.

5 RELATED WORK

Table 5 provides a summary of the core competencies of existing graph learning libraries. To briefly elaborate, recent libraries for

Table 4: Total training run time (seconds) over 100 epochs on MUTAG. Bold text refers to lowest time taken for training or are within error bounds of being the fastest.

Method	Original reference implementation	Only Geo2DR PyTorch modules	Geo2DR with compatible libraries Gensim/TensorFlow
DGK-WL	**3.06 ± 0.15**	3.33 ± 0.07	**3.19 ± 0.08**
DGK-SP	**6.95 ± 0.23**	**6.86 ± 0.27**	7.39 ± 0.08
DGK-GK	**9.46 ± 0.69**	19.41 ± 0.49	**9.89 ± 0.74**
Graph2Vec	**8.86 ± 0.05**	10.64 ± 0.11	**8.88 ± 0.06**
AWE-DD	1231.75 ± 21.81	**314.84 ± 8.91**	—

GCNN research such as GraphNets [1], DGL [26] and PyTorch Geometric [7] are characterised by a composite construction style of the message passing neural networks. Each method is constructed through the composition of convolution or pooling layers in the neural network and other preprocessing steps by the user. In contrast, graph kernel libraries such as GraKel [22] and GraphKernels [23], are API-oriented, with single line calls to specific implementations, where GraKel specifically follows the usage style of SciKit Learn [18] for compatibility. The recently released Karate Club [20] (its paper released the same week as Geo2DR) is an excellent API-oriented community detection and graph embedding library which implements several methods for distributed representations of graphs such as Graph2Vec and GL2Vec.

Geo2DR's underpinning design philosophy around composition of modules for method construction differentiates it from Karate Club. As stated in Section 3, the core focus is on the flexible yet rapid construction of methods with building blocks inspired by method creation in PyTorch and recent GCNN libraries. It allows a greater room for constructing novel methods in a modular fashion to encourage research and exploration. Ultimately, each of the

Table 5: This table is a simplified summary of core competencies of existing graph learning libraries. The column on method construction notes the style in which methods can be created. Composite refers to the creation of methods via composition of transformations, decompositions, neural network modules in the library by the user. On the other hand API refers to API-oriented "single-line" calls to specific implementations of methods, architectures etc. Composite/API* refers to few libraries exhibiting a composite design philosophy but have some single-line API calls to specific methods too popular to ignore such as the inclusion of Node2Vec in GEM and PyTorch Geometric but not as comprehensive as Karate Club or Geo2DR.

	Message passing network models	Graph Kernels	Distributed Representations of Graphs	Method Construction Style
GraphNets [1]	✓	×	×	Composite
DGL [26]	✓	×	×	Composite
GEM [9]	✓	×	×*	Composite/API*
Pytorch Geometric [7]	✓	×	×*	Composite/API*
Grakel [22]	×	✓	×	API
Graphkernels [23]	×	✓	×	API
Karate Club [20]	×	×	✓	API
Geo2DR	×	✓	✓	Composite

libraries cover specific competencies with their own usage philosophies, and we believe Geo2DR fills an important gap in supporting research of methods capable of learning distributed representations of graphs.

6 CONCLUSION

Through the characterisation of existing methods, and the reproduction of their results in Geo2DR, we have shown that the library is a successful amalgamation of the various components that enable learning distributed representations of graphs. Using the simple design methodology, one can quickly re-implement existing models, which is becoming an increasingly important part of reproducible research and designing novel architectures. By exploiting the modular structure and compatibility with other software and libraries the set of tools for constructing learning methods is broadened without having to deal with different data formats, language paradigms and workflows used by individual implementations. Using a host of re-implemented methods also allows for more homogenised experiment suites that can be used to more fairly compare existing and new methods in future research efforts. Geo2DR is now available with detailed documentation and examples as a starting point. The library will continue to evolve to add new components, compatibility with other libraries, tutorials, and accommodate new developments in the field.

ACKNOWLEDGEMENTS

Foremost, we would like to thank Dr. Yanardag [27], Dr. Narayanan [17] and Dr. Ivanov [11] for their correspondence, and making reference code available publicly. Furthermore we would like to thank the members of the AI Group at the Computer Laboratory for their patience and numerous proof readings of this work. Lastly, we would like to thank all the reviewers for their time and feedback to improve this work.

REFERENCES

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, and et al. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018). arXiv:1806.01261 http://arxiv.org/abs/1806.01261

[2] Mikhail Belkin and Partha Niyogi. 2001. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic* (Vancouver, British Columbia, Canada) *(NeurIPS'01)*. MIT Press, Cambridge, MA, USA, 585–591. http://dl.acm.org/citation.cfm?id=2980539.2980616

[3] Karsten M. Borgwardt and Hans-Peter Kriegel. 2005. Shortest-Path Kernels on Graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE Computer Society, Washington, DC, USA, 74–81. https://doi.org/10.1109/ICDM.2005.132

[4] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. 2005. Protein Function Prediction via Graph Kernels. *Bioinformatics* 21, 1 (2005), 47–56.

[5] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. 1991. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry* 34, 2 (01 Feb 1991), 786–797. https://doi.org/10.1021/jm00106a046

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Barcelona, Spain) *(NeurIPS'16)*. Curran Associates Inc., USA, 3844–3852. http://dl.acm.org/citation.cfm?id=3157382.3157527

[7] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[8] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 1263–1272. http://dl.acm.org/citation.cfm?id=3305381.3305512

[9] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78 – 94. https://doi.org/10.1016/j.knosys.2018.03.022

[10] Zellig S. Harris. 1954. Distributional Structure. *WORD* 10, 2-3 (1954), 146–162. https://doi.org/10.1080/00437956.1954.11659520

[11] Sergey Ivanov and Evgeny Burnaev. 2018. Anonymous Walk Embeddings. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 2191–2200. http://proceedings.mlr.press/v80/ivanov18a.html

[12] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. 2016. Benchmark Data Sets for Graph Kernels. Datasets available at http://graphkernels.cs.tu-dortmund.de.

[13] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (Toulon, France) *(ICLR'17)*.

[14] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (Beijing, China) *(ICML'14)*. JMLR.org, II–1188–II–1196. http://dl.acm.org/citation.cfm?id=3044805.3045025

[15] Enxhell Luzhnica, Ben Day, and Pietro Liò. 2019. On Graph Classification Networks, Datasets and Baselines. In *36th International Conference on Machine Learning* (Long Beach, USA) *(ICML'19)*.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

[17] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning Distributed Representations of Graphs. *CoRR* abs/1707.05005 (2017). arXiv:1707.05005

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, and et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[19] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50.

[20] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. arXiv:2003.04819 [cs.LG]

[21] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.* 12 (2011), 2539–2561.

[22] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. 2018. GraKeL: A Graph Kernel Library in Python. *arXiv preprint arXiv:1806.02193* (2018).

[23] Mahito Sugiyama, M. Elisabetta Ghisu, Felipe Llinares-López, and Karsten Borgwardt. 2017. graphkernels: R and Python packages for graph comparison. *Bioinformatics* 34, 3 (2017), 530–532. https://doi.org/10.1093/bioinformatics/btx602

[24] S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. 2010. Graph Kernels. *Journal of Machine Learning Research* 11 (2010), 1201–1242.

[25] Nikil Wale, Ian A. Watson, and George Karypis. 2008. Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. *Knowl. Inf. Syst.* 14, 3 (March 2008), 347–375. https://doi.org/10.1007/s10115-007-0103-5

[26] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, and et al. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019). https://arxiv.org/abs/1909.01315

[27] Pinar Yanardag and S.V.N. Vishwanathan. 2015. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD'15)*. ACM, New York, NY, USA, 1365–1374. https://doi.org/10.1145/2783258.2783417

[28] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montr&#233;al, Canada) *(NeurIPS'18)*. Curran Associates Inc., USA, 4805–4815. http://dl.acm.org/citation.cfm?id=3327345.3327389

A BRIEF PRIMER ON LEARNING DISTRIBUTED REPRESENTATIONS OF GRAPHS

Here we provide a brief and simplified primer on learning distributed representations of graphs. This will not fully describe the various intricacies of existing methods, but cover a conceptual framework common to almost all distributed representations of graphs particularly for learning representations of substructure patterns and whole graphs. Figure 2 is a diagrammatic representations of this conceptual framework.

Given a set of graphs $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_n\}$ one can induce discrete substructure patterns such as shortest paths, rooted subgraphs, graphlets, etc. using side-effects of algorithms such as the Floyd-Warshall or Weisfeiler-Lehman Graph Isomorphism test, and so on. This can be used to produce pattern frequency vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ describing the occurrence frequency of substructure patterns over a shared vocabulary $\mathbb{V}$. $\mathbb{V}$ is the set of unique substructure patterns induced across all of the graphs in the dataset $\mathbb{G}$.

Classically one may directly use these pattern frequency vectors within standard machine learning methods using vector inputs to perform some task. This is the approach taken by a variety of graph kernels [24, 27]. Unvfortunately, as the graphs of $\mathbb{G}$ and subtructure patterns induced become more complex through size or specificity, the number of induced patterns increases dramatically. This, in turn, causes the pattern frequency vectors of $\mathbf{X}$ to be extremely sparse and high-dimensional. The high specificity of the patterns and the sparsity of the pattern frequency vectors cause a phenomenon known as diagonal dominance across the kernel matrices wherein each graph becomes more similar to itself and dissimilar from others, degrading the classification performance [27].

To address this issue it is possible to learn dense and low dimensional distributed representations of graphs that are inductively biased to be similar when they contain similar substructure patterns and dissimilar when they do not. To achieve this, the construction of a corpus dataset $\mathcal{D}$ is required detailing the target-context relationship between a graph and its induced substructure as in our example or a substructure pattern to other substructure patterns. In the simplest form for graph-level representation learning one can implement $\mathcal{D}$ as tuples of graphs and substructure pattern $(\mathcal{G}_i, p_j) \in \mathcal{D}$ if $p_j \in \mathbb{V}$ and $p_j \in \mathcal{G}_i$.

The corpus is utilised with a method that incorporates Harris' distributive hypothesis [10] to learn the distributed representations of graphs. skipgram, cbow, PV-DM, PV-DBOW [14, 16] are a few examples of neural embedding methods that incorporate this inductive bias and are all present in the Geo2DR library. In skipgram with negative sampling, as used in Graph2Vec [17], the distributed representations can be learned by optimizing

$$\mathcal{L} = \sum_{\mathcal{G}_i \in \mathbb{G}} \sum_{p \in \mathbb{V}} |\{(\mathcal{G}_i, p) \in \mathcal{D}\}| (\log \sigma(\Phi_i \cdot \mathcal{S}_p) \\ + k \cdot \mathbb{E}_{p_N \in P_D} [\log \sigma(-\Phi_i \cdot p_N)]$$

over the corpus observations where $\Phi \in \mathbb{R}^{|\mathbb{G}| \times d}$ is the d dimensional matrix of graph embeddings we desire of the graph dataset $\mathbb{G}$, and Φ_i is embedding for $\mathcal{G}_i \in \mathbb{G}$. Similarly, $\mathcal{S} \in \mathbb{R}^{|\mathbb{V}| \times d}$ are the d dimensional embeddings of the substructure patterns in the vocabulary $\mathbb{V}$ so $\mathcal{S}_p$ represents the vector embedding corresponding to substructure pattern p. The embeddings of the substructure patterns are also tuned but ultimately not used, as we are interested in the graph embeddings in Φ. k is the number of negative samples with t_N being the sampled context pattern, drawn according to the empirical unigram distribution $P_D(p) = \frac{|\{p | \forall G_i \in \mathbb{G}, (G_i, p) \in \mathcal{D}\}|}{|D|}$.

The optimization of the above utility function creates the desired distributed representations of the targets in Φ, in this the case graph-level embeddings. These may be used as input for any downstream machine learning task and method that take vector inputs. The distributed representations benefit from having lower dimensionality than the pattern frequency vectors, in other words $|\mathbb{V}| >> d$, being non-sparse, and being inductively biased via the distributive

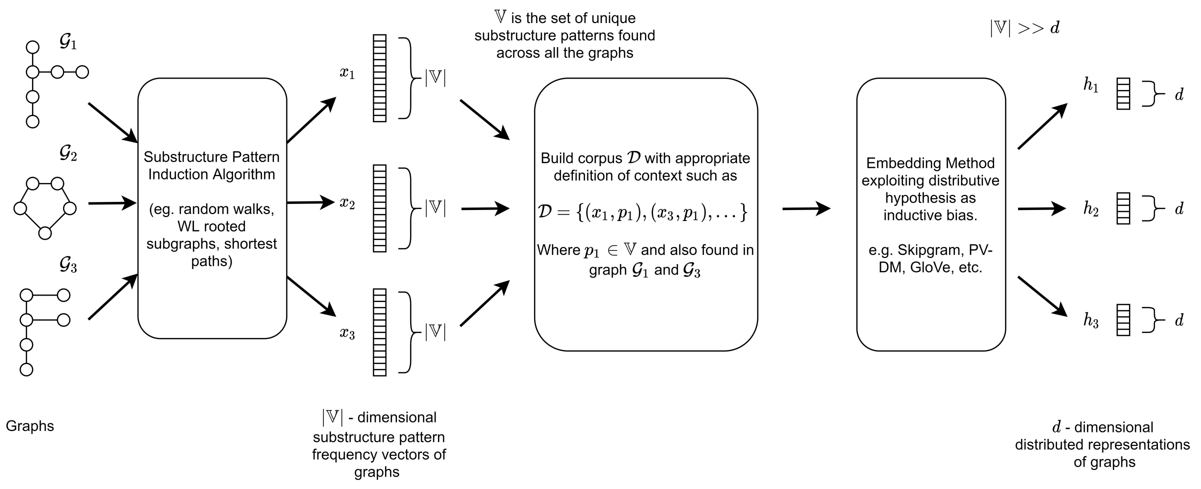
$\mathcal{G}_1$
$\mathcal{G}_2$
$\mathcal{G}_3$
Graphs
Substructure Pattern Induction Algorithm
(eg. random walks, WL rooted subgraphs, shortest paths)
x_1 $|\mathbb{V}|$
x_2 $|\mathbb{V}|$
x_3 $|\mathbb{V}|$
$|\mathbb{V}|$ - dimensional substructure pattern frequency vectors of graphs
$\mathbb{V}$ is the set of unique substructure patterns found across all the graphs
Build corpus $\mathcal{D}$ with appropriate definition of context such as
$\mathcal{D} = \{(x_1, p_1), (x_3, p_1), \dots\}$
Where $p_1 \in \mathbb{V}$ and also found in graph $\mathcal{G}_1$ and $\mathcal{G}_3$
Embedding Method exploiting distributive hypothesis as inductive bias.
e.g. Skipgram, PV-DM, GloVe, etc.
$|\mathbb{V}| >> d$
h_1 d
h_2 d
h_3 d
d - dimensional distributed representations of graphs

Figure 2: A conceptual framework for how methods for learning distributed representations of graphs are constructed, which guides the method design principles in Geo2DR.

hypothesis in an unsupervised manner. For more in-depth reading we recommend [10, 14, 16, 17, 27].

B CODE EXAMPLE

We present a construction of a simplified Graph2Vec model and training it to produce 32 dimensional distributed vector embeddings of the MUTAG graphs using Geo2DR modules. To start we need to download a dataset to study. We will use the well known MUTAG [5] downloaded from the TU Dortmund Graph Kernel Benchmark website [12]. Assume we have unpacked and saved the data into a directory called `org_data/` so the dataset as downloaded will be within the directory as `org_data/MUTAG/`.

Geo2DR uses the GEXF (Graph Exchange XML Format) as the permanent storage format for the graphs in a dataset. This is because it is compatible with network analysis software such as Gephi and NetworkX, and it is often useful to be able to study each graph individually; identified by a single file. Due to this design choice we need to transform the format of the downloaded dataset using tools available within the `data` subpackage as in the code sample below.

```
from geometric2dr.data import DortmundGexf

gexifier = DortmundGexf("MUTAG","org_data/","data/")
gexifier.format_dataset()
```

Listing 1: Formatting the downloaded dataset into GEXF format

This will result in the following dataset format:

- `data/MUTAG/` : a directory containing individual `.gexf` files of each graph. A graph will be denoted by the graph IDs used in the original data. In this case graph 0 would be `data/MUTAG/0.gexf`
- `data/MUTAG.Labels` : a plain-text file with each line containing a graph's file path and its classification label.

Given the preprocessed data we can now induce substructure patterns across the graph files. Here we will induce rooted subgraphs up to depth 2 using the Weisfeiler-Lehman node relabeling algorithm outlined in Shervashidze et al. [21].

```
from geometric2dr.decomposition.weisfeiler_lehman_patterns import
    wl_corpus
import geometric2dr.embedding_methods.utils as utils

dataset_path = "data/MUTAG"
graph_files = utils.get_files(dataset_path,".gexf")

wl_depth = 2
wl_corpus(graph_files,wl_depth)
```

Listing 2: Inducing rooted subgraphs across the graphs of the dataset

The `wl_corpus()` function induces rooted subgraph patterns across the list of `.gexf` files in `graph_files`, and builds a document for each graph describing the induced patterns within. These documents will have a special extension specific to each decomposition algorithm or can be set by the user. In this example the extension will be `.d2wl` to denote a Weisfeiler-Lehman decomposition to depth 2. Generating permanent files as a side effect of the graph decomposition process is useful for later study and also if we want to use the same induced patterns in the upcoming step of learning distributed representations of the graphs.

To learn distributed representations we need to construct a new target-context dataset. In Graph2Vec a graph is contextualised by the substructure patterns within it, and uses the PV-DBOW architecture with negative sampling to directly learn graph-level embeddings. Hence we use the `PVDBOWInMemoryCorpus` which is a extension of a standard `torch.utils.data.dataset` class. This can interface with a standard PyTorch dataloader to load the data into a `embedding_methods.skipgram` class that we train in a loop using a simple and recognizable `torch.nn` workflow.

```
import torch
import torch.optim as optim
from torch.utils.data import DataLoader
from geometric2dr.embedding_methods.pvdbow_data_reader import
    PVDBOWInMemoryCorpus
from geometric2dr.embedding_methods.skipgram import Skipgram

# Instantiate corpus dataset, dataloader and skipgram
# architecture
corpus = PVDBOWCorpus(dataset_path,".d2wl")
dataloader = DataLoader(corpus, batch_size=1000, shuffle=False,
    collate_fn=corpus.collate)
skipgram = Skipgram(num_targets=corpus.num_graphs, vocab_size=
    corpus.num_subgraphs, emb_dimension=32)

optimizer = optim.SGD(skipgram.parameters(), lr=0.1)
for epoch in range(100):
  for i, sample_batched in enumerate(dataloader):
    if len(sample_batched[0]) > 1:
      pos_target = sample_batched[0]
      pos_context = sample_batched[1]
      neg_context = sample_batched[2]

      optimizer.zero_grad()
      loss = skipgram.forward(pos_target, pos_context, neg_context)
      loss.backward()
      optimizer.step()

final_graph_embeddings = skipgram.target_embeddings.weight
```

Listing 3: Creating a target-context dataset then attaching a dataloader that feeds the corpus data into a skipgram model and training it.

The completion of the training provides the final graph embeddings. As this is such a common proces, Geo2DR also comes with a number of `Trainer` classes which build corpus datasets, loaders, train neural language models, and save their outputs. All of the code above can be replaced with this short trainer.

```
from geometric2dr.embedding_methods.pvdbow_trainer import
    InMemoryTrainer

trainer = InMemoryTrainer(corpus_dir=dataset_path, extension=".d2wl
    ", output_fh="graph_embeddings.json", emb_dimension=32,
    batch_size=1000, epochs=100, initial_lr=0.1, min_count=0)
trainer.train()
final_graph_embeddings = trainer.skipgram.give_target_embeddings()
```

Listing 4: Trainer example of performing all of listing 1.3

C SUPPLEMENTARY: DATASET DETAILS

Table 6 contains descriptive information about each of the datasets as they were used within the empirical evaluation described in Section 4 of the main paper. All of the datasets are commonly used benchmark datasets downloaded from Kersting et al.'s [12] repository[2]. After downloading the datasets they were processed into the format used by Geo2DR with the included data formatter. In each of the datasets the discrete node labels are exposed, but not the edge labels. For unlabelled datasets such as REDDIT-B and IMDB-M, the nodes are labelled by their degree as in Shervashidze et al. [21] to enable methods such as the WL rooted subgraph decomposition to induce patterns in the graphs. This was also applied to methods which can directly handle unlabelled graphs for conformity.

The graphs come from a variety of contexts and domains. MUTAG, ENZYMES and PROTEINS are datasets which have their roots in bioinformatics research. The graphs within them represent molecules with nodes representing atoms and edges denoting chemical bonds or spatial proximity between different atoms.

[2]ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets

Graph labels describe different properties of the molecules such as mutagenicity or whether a protein is an enzyme. NCI1 is a chemoinformatics dataset describing compounds screened for their ability to surpress or inhibit the growth of a panel of human tumor cell lines. REDDIT-B and IMDB-M are social network based datasets. In REDDIT-B each graph corresponds to an online discussion thread where nodes correspond to users, and there is an edge between the nodes if at least one responded to another's comment. IMDB-M is a movie collaboration dataset where each graph corresponds to an ego-network of an actor or actress.

D SUPPLEMENTARY: HYPERPARAMETER SELECTIONS OF RE-IMPLEMENTED METHODS

For each of the methods described in Section 4 we prescribed a grid search over the following hyper-parameter settings inspired by the settings of the original papers:

D.1 Graph Kernels

- **WL Rooted Subgraphs:** Rooted subgraphs up to depth 2 induced.
- **Shortest Paths:** Shortest paths of all pairs of nodes induced.
- **Graphlets:** Graphlets of size 7 induced, sampling 100 graphlets per graph.
- **Anonymous Walks:** Anonymous walks of length 10 induced exhaustively from each node in the graph.

D.2 Deep Graph Kernels and Graph Embeddings

- **DGK-WL:** Rooted subgraphs of up to depth 2 induced. Trained Skipgram model with negative sampling using 10 negative samples with an Adam optimiser for 5 and 100 epochs using batch sizes of 10000 and 1000 with an initial learning rate of 0.1 and 0.01 adjusted by a cosine annealing scheme. Substructure embedding sizes of 2, 5, 10, 25, 50 dimensions were generated. Graph kernels were constructed using the formulation described in Yanardag and Vishwanathan [27].
- **DGK-SP:** Shortest paths of all pairs of nodes induced. Trained Skipgram model with negative sampling using 10 negative samples with an Adam optimiser for 5 and 100 epochs using batch sizes of 10000 and 1000 with an initial learning rate of 0.1 and 0.01 adjusted by a cosine annealing scheme. Substructure embedding sizes of 2, 5, 10, 25, 50 dimensions were generated. Graph kernels were constructed using the formulation described in Yanardag and Vishwanathan [27].
- **DGK-GK:** Graphlets of size 7 induced, sampling 2, 5, 10, 25, and 50 graphlets for each graph. Trained Skipgram model with negative sampling using 10 negative samples with an Adam optimiser for 5 and 100 epochs using batch sizes of 10000 and 1000 with an initial learning rate of 0.1 and 0.01 adjusted by a cosine annealing scheme. Substructure embeddings of 2, 5, 10, 25, 50 dimensions were generated. Graph kernels were constructed using the formulation described in Yanardag and Vishwanathan [27].

Table 6: Descriptive information about datasets used in the experimental evaluation. N refers the number of graphs in the datasets. C is the number of graph classification labels. Avg. Nodes and Avg. Edges denote the average number of nodes and edges found in the graphs of the dataset respectively. Finally Node Labels indicates whether the nodes are discretely labelled. The * refers to datasets which originally do not have node labels, but are subsequently labelled by their degree as described in Shervashidze et al. [21]

Dataset	N	C	Avg. Nodes	Avg. Edges	Node Labels
MUTAG [5]	188	2	17.93	19.79	Yes
ENZYMES [4]	600	6	32.63	62.14	Yes
PROTEINS [4]	1113	2	39.06	72.82	Yes
NCI1 [25]	4110	2	29.87	32.3	Yes
REDDIT-B [27]	2000	2	429.63	497.75	No*
IMDB-M [27]	1500	3	13	65.94	No*

- **Graph2Vec:** Rooted subgraphs of up to depth 2 induced. Trained over PV-DBOW (Skipgram) model with negative sampling using 10 negative samples with an Adam optimiser for 25, 50, 100 epochs and batch sizes of 512, 1024, 2048, 10000 with an initial learning rate of 0.1 adjusted by a cosine annealing scheme. Graph embeddings of 128 and 1024 dimensions were learned.
- **AWE-DD:** Anonymous walks of length 10 induced exhaustively. Trained over PV-DM architecture with negative sampling using 10 negative samples with an Adagrad optimiser (as in reference implementation) for 100 epochs with batch sizes 100, 500, 1000, 5000, 10000 with an initial learning rate of 0.1. Window-sizes of 4, 8, 16 were used to extract context anonymous walks around the target anonymous walk in the PV-DM architecture.