DOI: 10.14754/CEU.2024.08

Essays in Machine Learning and Networks

by

Olivér Kiss

Submitted to

Central European University

Department of Economics and Business

In partial fulfilment of the requirements for the degree of

Doctor of Philosophy in Economics

Supervisor: Ádám Szeidl

Budapest, Hungary - Vienna, Austria

DOI: 10.14754/CEU.2024.08

Copyright Notice and Attestation

Author: Olivér Kiss

Title: Essays in Machine Learning and Networks

Degree: Ph.D.

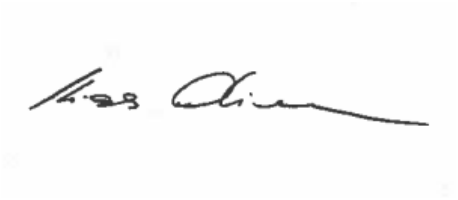
Dated: June 25, 2024

Copyright notice:

Attestation:

Hereby I testify that this thesis contains no material accepted for any other degree in any other institution and that it contains no material previously written and/or published by another person except where appropriate acknowledgement is made.

Signature of the author:



DOI: 10.14754/CEU.2024.08

Co-author contribution

Chapter 1: Machine Learning on Networks

Joint work with Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar

This chapter consists of three published papers. In the case of Karate Club and Little Ball of Fur, I contributed the experimental evaluations and developed a minor part of the Python libraries while Benedek Rozemberczki did most of the software development and wrote the theoretical backgrounds for the papers. For Pytorch Geometric Temporal, Benedek Rozemberczki wrote the majority of the software, Paul Scherer ran the experiments, and I and other co-authors collected and contributed new datasets for the experimental evaluations.

Chapter 2: The Shapley Value in Machine Learning

Joint work with Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Sebastian Nilsson, and Rik Sarkar

The original idea for the paper came from Benedek Rozemberczki. I contributed the majority of the theoretical background on the Shapley Value. The work on the approximation techniques was mostly done by Lauren Watson. Other authors contributed to individual games described in the paper according to their research areas and expertise.

DOI: 10.14754/CEU.2024.08

Abstracts

Chapter 1: Machine Learning on Networks[1].

Joint work with Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar

This chapter consists of three published papers, all developing and examining machine learning tools for graph-structured data.

Graphs encode important structural properties of complex systems. Machine learning on graphs has therefore emerged as an important technique in research and applications. We present *Karate Club* – a Python framework combining more than 30 state-of-the-art graph mining algorithms. We show *Karate Club*'s efficiency in learning performance on a wide range of real-world clustering problems and classification tasks along with supporting evidence of its competitive speed.

Sampling graphs is an important task in data mining. We propose *Little Ball of Fur* a Python library that includes more than twenty graph sampling algorithms. Our experiments demonstrate that *Little Ball of Fur* can speed up node and whole graph embedding techniques considerably while only mildly deteriorating the predictive value of distilled features.

We present PyTorch Geometric Temporal, a deep learning framework combining state-of-the-art machine learning algorithms for neural spatiotemporal signal processing. Experiments demonstrate the predictive performance of the models implemented in the library on real-world problems such as epidemiological forecasting, ride-hail demand prediction, and web traffic management. Our sensitivity analysis of runtime shows that the framework can potentially operate on web-scale datasets with rich temporal features and spatial structure.

[1]Papers in this chapter were published in the Proceedings of the 29th ACM International Conference on Information and Knowledge Management (Rozemberczki, Kiss and Sarkar, 2020*b*,*a*) and the Proceedings of the 30th ACM International Conference on Information and Knowledge Management (Rozemberczki et al., 2021*b*)

DOI: 10.14754/CEU.2024.08

Chapter 2: The Shapley Value in Machine Learning[2]

Joint work with Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Sebastian Nilsson, and Rik Sarkar

Over the last few years, the Shapley value, a solution concept from cooperative game theory, has found numerous applications in machine learning. In this paper, we first discuss fundamental concepts of cooperative game theory and axiomatic properties of the Shapley value. Then, we give an overview of the most important applications of the Shapley value in machine learning: feature selection, explainability, multi-agent reinforcement learning, ensemble pruning, and data valuation. We examine the most crucial limitations of the Shapley value and point out directions for future research.

Chapter 3: Peer Effects in Multiplex Networks

How social and economic networks govern real-life phenomena has been studied extensively in the past decades. While in some settings constructing randomized experiments is possible, most social ties are difficult or impossible to randomize. In such cases, the estimation of peer effects must rely on observational data. Such estimation is, however, hindered by a range of difficulties posed by the specific data structure. This paper proposes using spatial models to address the problem of omitted variable bias caused by correlated social networks and to disentangle peer effects in more complex social situations. The validity of the approach is verified using Monte Carlo simulations and synthetic graphs. The difference arising from the proposed simultaneous peer effects estimation is further illustrated using data from a technology adoption field experiment in Uganda.

[2]This paper was published in the Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (Rozemberczki et al., 2022).

DOI: 10.14754/CEU.2024.08

Acknowledgements

I am indebted to my advisor, Ádám Szeidl for providing me with guidance and valuable feedback throughout my years at CEU. His advising strongly influenced my thinking about economics and research. It has been a privilege to work with him. I would also like to thank all CEU faculty members who have supported me throughout this journey.

I was also fortunate to work with exceptional colleagues and friends during my studies. I am grateful to Benedek Rózemberczki for all the projects we worked on together and for his encouragement in times when I needed motivation.

I thank the examiners of this thesis, Brendan Lucier and Miklós Koren for their in-depth reviews and suggestions. Their constructive feedback improved this thesis greatly.

I would also like to express my gratitude to the staff members of the Department of Economics and Business. Their limitless efforts to minimize the bureaucratic burden on students and faculty made CEU a better place for everyone.

Finally, I would like to thank my friends and my family. Their support and love means the world to me.

DOI: 10.14754/CEU.2024.08

Contents

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

List of Tables

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

List of Figures

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

Introduction

This thesis consists of two co-authored chapters and one single-authored chapter, all focusing on machine learning and/or social and economic networks.

The first chapter, titled Machine Learning on Networks consists of three published papers on applied network analysis tools. Karate Club is a python library implementing unsupervised learning methods for graph structured data. It implements over 30 algorithms that practitioners can use to perform data mining tasks on graphs. The paper has been published in the Proceedings of the 29th ACM International Conference on Information and Knowledge Management, a major double-blind-reviewed venue for resource papers, and it is widely used by the relevant research community. In May 2024, the library has been downloaded from the Python Package Index 4,379 times[3]. The methods implemented in this paper cover three relevant unsupervised problems: (i) community detection, (ii) node embeddings and (iii) graph embeddings. Community detection algorithms are used to identify groups if nodes in a network that are either densely interconnected or share similar node characteristics. Node embeddings aim to assign a low dimensional vector to each node in a network (embed them in a low dimensional space) in a way that their distance in this low dimensional space reflects a notion of similarity. For example, neighborhood-based node embeddig techniques aim assign this lower dimensional vector in a way such that nodes that have a lower network distance are closer in this lower dimensional space too. These embeddings can be used as controls in numerous downstream estimation and forecasting tasks. Graph embeddings are very similar, but instead of embedding nodes, they assign low

[3]According to PyPI Download Stats: `https://pypistats.org/packages/karateclub`

DOI: 10.14754/CEU.2024.08

dimensional vector representations to whole graphs in a dataset consisting of a large set of networks. Intuitively, the extracted information can be used in downstream machine learning and regression models where each observation is a network.

The second resource paper, Little Ball of Fur, provides graph sampling methods for the research community. Just like Karate Club, it has been published in the Proceedings of the 29th ACM International Conference on Information and Knowledge Management. In May 2024, the library has been downloaded from the Python Package Index 145 times[4]. This paper also provides a set of evaluations for the implemented graph sampling techniques, which is valuable to researchers relying on graph sampling. Sampling networks is not an evident problem for economists studying spillover effects in economic and social networks. Random sampling of nodes or edges can destroy salient information encoded in networks. Applying the correct sampling method to retain the relevant information is essential. Our research provides insights on how different sampling techniques perform in terms of preserving a set of ground truth statistics.

Pytorch Geometric Temporal has been published a year later in the Proceedings of the Proceedings of the 30th ACM International Conference on Information and Knowledge Management, the same double-blind-revieved venue. It has been downloaded from the Python Package Index 1,919 times in May 2024[5]. This library implements a wide set of tools that can be used in spatiotemporal machine learning models. These models are used to describe dynamics when agents in a social or economic network are observed over time. It covers three major settings: (i) when the network is static, but the characteristics of the agents evolve over time, (ii) when the characteristics of the agents are fixed but the network describing their connections changes over time and (iii) when both the network and the attributes of the agents change over time. These models are primarily used in predictive tasks, and can be applied in a wide variety of problems relevant to many research disciplines. Among others, spread of infectious diseases in space

[4]According to PyPI Download Stats: `https://pypistats.org/packages/littleballoffur`
[5]According to PyPI Download Stats: `https://pypistats.org/packages/torch-geometric-temporal`

DOI: 10.14754/CEU.2024.08

and time, dissemination of information over social networks, renewable energy production, or travels in public transport networks are all spatiotemporal processes that can be modelled using our library.

Chapter 2, titled The Shapley Value in Machine Learning provides a comprehensive overview of how the Shapley Value - a solution concept in cooperative game theory - can be applied in the domain of machine learning to assign values to different components in machine learning models or frameworks. This paper was published in the Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, a double-blind-reviewed venue. To our knowledge, this is the first paper that provides an extensive set of formal definitions for different machine learning games.

Chapter 3, titled Peer Effects in Multiplex Networks contributes to the literature on peer effects. It proposes a model that incorporates multiple types of connections (e.g.: friendships, family relations, etc.) among the same set of agents simultaneously and discusses how such a model can be estimated using spatial autoregressive models with multiple weight matrices. This chapter provides both simulated and empirical evidence that failing to account for multiple types of networks results in biased peer effect estimates. Results show that researchers relying on observational data for peer effect estimations are likely to estimate significantly biased peer effects unless they control for all relevant networks simultaneously.

DOI: 10.14754/CEU.2024.08

Chapter 1

Machine Learning on Networks

Joint work with Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar

This chapter consists of three papers on machine learning tools developed to extract information from graph-structured data. Section 1.1 presents Karate Club, a Python library with unsupervised learning methods for static graphs. Section 1.2 presents Little Ball of Fur, a package implementing an extensive set of graph sampling algorithms. Section 1.3 discusses Pytorch Geometric Temporal, a spatiotemporal signal processing extension for Pytorch. All of these papers discuss the design principles behind the libraries and provide experimental evaluations of their performances.

DOI: 10.14754/CEU.2024.08

1.1 Karate Club: An API Oriented Open-Source Python Framework for Unsupervised Learning on Graphs

1.1.1 Introduction

Techniques that extract features from graph data in an unsupervised manner (Perozzi, Al-Rfou and Skiena, 2014; Yang and Leskovec, 2013; Narayanan et al., 2017) have seen an increasing success in the machine learning community. Features automatically extracted by these methods aim to retain information in a lower dimensional space based on similarity metrics. The vast majority of these algorithms embed neighborhood information (nodes with a lower distance are closer in the embedding space), structural information (nodes that have similar structural characteristics, like degree or centrality, are closer in the embedding space) or attribute information (nodes that are characterized by similar attributes will be closer in the embedding space). These embeddings can serve as inputs for link prediction, node and graph classification, community detection and various other tasks tasks (Rozemberczki, Allen and Sarkar, 2019; Perozzi, Al-Rfou and Skiena, 2014; Narayanan et al., 2017; Perozzi et al., 2017; Yanardag and Vishwanathan, 2015) in a wide range of real world research and application scenarios. Graph mining tools such as SNAP, NetworkX or GraphTool (Leskovec and Krevl, 2014; Hagberg, Swart and S Chult, 2008; Peixoto, 2014) have contributed to enhancement and development of machine learning pipelines. The need for complicated custom feature engineering is reduced by unsupervised graph mining techniques. This approach produces features that are naturally reusable on multiple types of tasks. Recent research (Perozzi, Al-Rfou and Skiena, 2014; Perozzi et al., 2017; Tsitsulin et al., 2018) has made such feature extraction highly efficient and parallelizable.

The democratization of machine learning for tabular data was led by frameworks which made fast paced development possible. Tools such as Scikit, TensorFlow or Pytorch (Pedregosa et al., 2011; Buitinck et al., 2013; Abadi et al., 2016; Paszke et al., 2019; Rozemberczki, Kiss and

DOI: 10.14754/CEU.2024.08

Sarkar, 2020*b*) are available in general purpose scripting languages with easy to use consistent interfaces. On the other hand, current graph based learning frameworks are less mature and of limited scope. NetworkX and GraphTool (Hagberg, Swart and S Chult, 2008; Peixoto, 2014), for example, host certain community detection algorithms, but none for whole graph or node embedding. In addition, some of these tools (e.g.: SNAP, GraphTool Leskovec and Krevl (2014); Peixoto (2014)) have significant barriers for the end users in terms of installing prerequisites and custom data structures used for representing graphs.

We propose *Karate Club*, an open source Python framework for unsupervised learning on graphs. Our work implements a wide range of methods that can be used in three unsupervised learning scenarios: community detection, node embeddings for node classification, and graph embeddings for graph classification. We implemented *Karate Club* with consistent API oriented design principles in mind which makes the library end user friendly and modular. The name of the package is inspired by Zachary's Karate Club (Zachary, 1977) – a network commonly used to demonstrate network algorithms. The design of this machine learning toolbox was motivated by the principles used to create the widely used *scikit-learn* package (Buitinck et al., 2013).

The design entails a number of fundamental engineering patterns. Each algorithm has a sensible default hyperparameter setting which helps non expert practitioners. To further ease the use of our package, algorithms have a limited number of shared, publicly available methods (e.g. fit). Models ingest data structures from the scientific Python ecosystem (Hagberg, Swart and S Chult, 2008; Virtanen et al., 2019; Walt, Colbert and Varoquaux, 2011) as input and the generated output also follows these formats. The inner model mechanics are always implemented as private methods using optimized back-end libraries such as NumPy, SciPy, Pysgp or GenSim (Walt, Colbert and Varoquaux, 2011; Virtanen et al., 2019; Defferrard et al., 2017; Rehurek and Sojka, 2011) for computing. These principles combined with the extensive documentation ensure that *Karate Club* is accessible to a wider audience than the currently available open-source graph mining frameworks.

Our empirical evaluation focuses on three common graph mining tasks: non-overlapping

DOI: 10.14754/CEU.2024.08

community detection, node and graph classification. We compare the learning performance of node and graph level algorithms implemented in our framework on various real world social, web and collaboration networks (collected from Deezer, Reddit, Facebook, Twitch, Wikipedia and GitHub). With respect to the runtime, models in *Karate Club* show reasonable scalability which we demonstrate by the use of synthetic data.

Our contributions. Specifically our work makes the following key contributions:

1. We release *Karate Club*, a Python graph mining framework which provides a wide range of easy to use community detection, node and whole graph embedding procedures.

2. We demonstrate with code the main ideas of the API oriented framework design: hyperparameter encapsulation and inspection, available public methods, dataset ingestion, output generation, and interfacing with downstream learning algorithms and evaluation methods.

3. We evaluate the learning performance of the framework on real world community detection, node and graph classification problems. We validate the scalability of the algorithms implemented in our framework.

4. We open sourced with the framework a detailed documentation and released multiple large graph classification datasets.

The remainder of this paper is structured as follows. In Section 1.1.3 we discuss the covered graph mining techniques. We overview the main principles behind *Karate Club* in Section 1.1.4 where we included detailed examples to explain these design ideas. The learning performance and scalability of the algorithms in the package are evaluated in Section 1.1.5. We review related data mining libraries in Section 1.1.2. The paper concludes with Section 1.1.6 where we discuss future directions. The source code of *Karate Club* can be downloaded from `https://github.com/benedekrozemberczki/karateclub`; the Python

DOI: 10.14754/CEU.2024.08

package can be installed from the *Python Package Index*. Extensive documentation is available at `https://karateclub.readthedocs.io/en/latest/` with detailed examples.

1.1.2 Related Work

In this section we discuss how the design of our framework is related to existing machine learning frameworks, and what differentiates it from other graph mining tools.

API oriented machine learning frameworks

Scikit-learn (Pedregosa et al., 2011; Buitinck et al., 2013) is a machine learning framework with a consistent and easy-to-use design. The *scikit-learn* models are characterised by models with a consistent API, their constructors have encapsulated sensible hyperparameters and utilize widely used Python data structures for data ingestion and output generation. This compositional design of the framework results in a low number of model classes and reusable model blocks and enables fast deployment. The *Karate Club* API draws heavily from the ideas of *scikit-learn* and the output generated by *Karate Club* is suitable as input for *scikit-learn*'s machine learning procedures. While our general design patterns are similar to that of Scikit-learn, it is important to note that we do not rely on any design frameworks or API components from any other library and our tool implements these internally.

Graph mining libraries

The *Karate Club* framework is differentiated from other graph mining libraries because of lightweight prerequisites and wide coverage of the learning techniques. First, the *SNAP* and *GraphTool* packages both have C++ prerequisites which have to be pre-compiled and installed. Our framework only has Python dependencies and builds on top of the *NetworkX* project. Second, the *SNAP* (Leskovec and Krevl, 2014) library only covers specific methods which were created by the authors of the framework. The *NetworkX* (Hagberg, Swart and S Chult, 2008) and

DOI: 10.14754/CEU.2024.08

GraphTool (Peixoto, 2014) libraries only provide community detection tools. Node and whole graph embedding is not supported by these frameworks.

1.1.3 Graph mining procedures in *Karate Club*

In this section, we briefly discuss the various graph mining algorithms which are available in the 1.0. release of the *Karate Club* package.

Community detection

Community detection techniques cluster the vertices of the graph into densely connected groups of nodes. This grouping can be the final result or an input for a downstream learning algorithm (e.g. node classification or link prediction).

Karate Club currently contains several methods for overlapping and non-overlapping community detection. Non-overlapping community detection is analogous to clustering, and assumes that a node can only belong to a single group; see, for example, Li et al. (2019*b*); Raghavan, Albert and Kumara (2007); Prat-Pérez, Dominguez-Sal and Larriba-Pey (2014); Rozemberczki et al. (2019). While overlapping community detection is analogous to fuzzy clustering as nodes have an affiliation with multiple clusters (e.g.: Yang and Leskovec (2013); Ye, Chen and Zheng (2018); Wang et al. (2017); Sun et al. (2017); Kuang, Ding and Park (2012)).

Node embedding

Node embeddings map vertices of a graph into an Euclidean space in which those that are deemed to be similar according to a certain notion will be in close proximity. The Euclidean representation makes it easier to apply standard machine learning libraries for node classification, link prediction, clustering etc.

Neighbourhood preserving embedding maintains the proximity of nodes in the graph when an embedding is created. These methods implicitly (Perozzi, Al-Rfou and Skiena, 2014; Perozzi

DOI: 10.14754/CEU.2024.08

et al., 2017; Rozemberczki and Sarkar, 2018) or explicitly (Cao, Lu and Xu, 2015; Jundong Li, 2019; Sun and Fevotte, 2014; Qiu et al., 2018) decompose a proximity matrix (e.g. powers of the adjacency matrix or a sum of these matrices) to create the node embedding.

Structural embedding conserves the structural roles of nodes in the embedding space (Henderson et al., 2012; Ahmed et al., 2019; Donnat et al., 2018). Nodes with similar embeddings have a similar distribution of descriptive statistics (e.g. centrality and clustering coefficient) in their vicinity. Embeddings are distilled by the decomposition of matrices representing structural features of nodes.

Attributed embedding retains the neighbourhood structure and generic feature similarity of nodes when the embedding is learned. This learning involves the joint factorization of the node-node and node-feature matrices with a direct (Yang et al., 2018, 2015) or implicit matrix decomposition technique (Rozemberczki, Allen and Sarkar, 2019; Zhang et al., 2018).

Meta embedding combines information from neighbourhood preserving, structural and attributed embeddings in order to create higher representation quality embeddings (Yang et al., 2017).

Whole graph embedding and summarization

Whole graph embedding and summarization techniques create fixed size representations of entire graphs as points in a Euclidean space. Those graphs which are close in the embedding space share structural patterns such as subtrees. These representations are used for a range of graph level tasks – graph classification, regression and whole graph clustering.

Spectral fingerprints extract statistics from the eigenvectors and eigenvalues of the graph Laplacian (Tsitsulin et al., 2018; de Lara and Edouard, 2018; Verma and Zhang, 2017). Vectors of the descriptive statistics are used as the whole graph representation.

Implicit factorization techniques create a graph – structural feature matrix (Narayanan et al., 2017; Chen and Koga, 2019) by enumerating string features in the graphs. This matrix is decomposed in order to create whole graph descriptors and feature embeddings jointly.

DOI: 10.14754/CEU.2024.08

1.1.4 Design Principles

When we created *Karate Club*, we used an API-oriented machine learning system design point of view (Pedregosa et al., 2011; Buitinck et al., 2013) in order to make an end-user-friendly machine learning tool. This API-oriented design principle entails a few simple ideas. In this section, we discuss these ideas and their apparent advantages with appropriate illustrative examples in great detail.

Encapsulated model hyperparameters and inspection

An unsupervised *Karate Club* model instance is created by using the constructor of the appropriate Python object. This constructor has a *default hyperparameter setting* which allows for sensible out-of-the-box model usage. In simple terms, this means that the end user does not need to understand the inner model mechanics in great detail to use the methods implemented in our framework. We set these default hyperparameters to provide a reasonable learning and runtime performance. If needed, these model hyperparameters can be modified at the model instance creation time with the appropriate re-parametrization of the constructor. The hyperparameters are stored as *public attributes* to allow the inspection of model settings.

We demonstrate the encapsulation of hyperparameters by the code snippet in Figure 1.1. First, we want to create an embedding for a *NetworkX* generated Erdos-Renyi graph (line 4) with the standard hyperparameter settings. When the model is constructed and fitted (lines 6-7) we do not change default hyperparameters and we can print the standard setting of the dimensions hyperparameter (line 8). Second, we decided to set a different number of dimensions, so we created and fitted a new model (lines 10-11) and we print the new value of the dimensions hyperparameter (line 12).

DOI: 10.14754/CEU.2024.08

```
import networkx as nx
from karateclub import DeepWalk

graph = nx.gnm_random_graph(100, 1000)

model = DeepWalk()
model.fit(graph)
print(model.dimensions)

model = DeepWalk(dimensions=64)
model.fit(graph)
print(model.dimensions)
```

Figure 1.1: Creating a synthetic graph, using a DeepWalk model with standard and modified hyperparameter settings.

API Consistency and non-proliferation of classes

Each unsupervised machine learning model in *Karate Club* is implemented as a separate class that inherits from the *Estimator* class. Algorithms implemented in our framework have a limited number of *public methods* as we do not assume that the end user is particularly interested in the algorithmic details related to a specific technique. All models are trained by the use of the *fit* method which takes the inputs (graph, node features) and calls the appropriate private methods to learn an embedding or clustering. Node and graph embeddings are returned by the *get_embedding* public method and cluster memberships are retrieved by calling *get_memberships*.


```
import networkx as nx
from karateclub import DeepWalk

graph = nx.gnm_random_graph(100, 1000)

model = DeepWalk()
model.fit(graph)
embedding = model.get_embedding()
```

Figure 1.2: Creating a synthetic graph, using the DeepWalk constructor, fitting the embedding and returning it.

We avoided the proliferation of classes with two specific strategies. First, the inputs used by our framework and the outputs generated do not rely on custom data classes. This helps to prevent the unnecessary growth of the number of classes and also helps with interfacing with downstream applications. Second, algorithms that use the same data pre-processing or algorithmic step (e.g. truncated random walk, Weisfeiler-Lehman hashing) were built on shared blocks.

In Figure 1.2 we create a random graph (line 4), and a DeepWalk model with the default hyperparameters (line 6), we fit this model (line 7) using the public *fit* method and return the embedding by calling the public *get_embedding* method (line 8).

```
1 import networkx as nx
2 from karateclub import Walklets
3
4 graph = nx.gnm_random_graph(100, 1000)
5
6 model = Walklets()
7 model.fit(graph)
8 embedding = model.get_embedding()
```

Figure 1.3: Creating a synthetic graph, using the Walklets constructor, fitting the embedding and returning it.

The example in Figure 1.2 can be modified to create a *Walklets* embedding with minimal effort by changing the model import (line 2) and the constructor (line 6) – these modifications result in the snippet of Figure 1.3.

Looking at these two snippets the advantage of the API-driven design is evident as we only needed a few modifications. First, we had to change the import of the embedding model. Second, we needed to modify the model construction, and the default hyperparameters were already set. Third, the public methods provided by the *DeepWalk* and *Walklets* classes behave the same way. An embedding is learned with *fit* and it is returned by *get_embedding*. This allows for quick and minimal changes to the code when an upstream unsupervised model used for feature extraction performs poorly.

DOI: 10.14754/CEU.2024.08

Standardized dataset ingestion

We designed *Karate Club* to use standardized dataset ingestion when a model is fitted. Practically this means that algorithms that have the same purpose use the same data types for model training. In detail:

- Neighbourhood-based and structural node embedding techniques use a single *NetworkX* graph as input for the fit method.
- Attributed node embedding procedures take a *NetworkX* graph as input and the features are represented as a *NumPy* array or as a *SciPy* sparse matrix. In these matrices, rows correspond to nodes and columns to features.
- Graph level embedding methods and statistical graph fingerprints take a list of *NetworkX* graphs as input.
- Community detection methods use a *NetworkX* graph as an input.

High performance model mechanics

The underlying mechanics of the graph mining algorithms were implemented using widely available Python libraries which are not operation system dependent and do not require the presence of other external libraries like *TensorFlow* or *PyTorch* does (Abadi et al., 2016; Paszke et al., 2019). The internal graph representations in *Karate Club* use *NetworkX*. Dense linear algebra operations are carried out using *NumPy* and their sparse counterparts use *SciPy*. Implicit matrix factorization techniques (Perozzi, Al-Rfou and Skiena, 2014; Perozzi et al., 2017; Rozemberczki, Allen and Sarkar, 2019; Ahmed et al., 2019; Zhang et al., 2018) utilize the *GenSim* (Rehurek and Sojka, 2011) package to and methods which rely on graph signal processing use *PyGSP* (Defferrard et al., 2017). It is important to note that these libraries only provide fast and/or standardized ways for storing the raw data or performing mathematical

DOI: 10.14754/CEU.2024.08

transformations on it. All internal model calculations are implemented by us, with the exception of topic modeling algorithms provided by *GenSim* (Rehurek and Sojka, 2011).

```
import community
import networkx as nx
from karateclub import LabelPropagation, SCD

graph = nx.gnm_random_graph(100, 1000)

model = SCD()
model.fit(graph)
scd_memberships = model.get_memberships()

model = LabelPropagation()
model.fit(graph)
lp_memberships = model.get_memberships()

print(community.modularity(scd_memberships, graph))
print(community.modularity(lp_memberships, graph))
```

Figure 1.4: Creating a synthetic graph, clustering with two community detection techniques and using an external library to evaluate the modularity of clusterings.

Standardized output generation and downstream interfacing

The standardized output generation of *Karate Club* ensures that unsupervised learning algorithms which serve the same purpose always return the same type of output with a consistent data point ordering. There is a very important consequence of this design principle. When a certain type of algorithm is replaced with the same type of algorithm, the downstream code which uses the output of the upstream unsupervised model does not have to be changed. Specifically the outputs generated with our framework use the following data structures:

- *Node embedding algorithms* (neighbourhood preserving, attributed and structural) always return a *NumPy* float array when the *get_embedding* method is called. The number of rows in the array is the number of vertices and the row index always corresponds to the vertex index. Furthermore, the number of columns is the number of embedding dimensions.

DOI: 10.14754/CEU.2024.08

- *Whole graph embedding methods* (spectral fingerprints, implicit matrix factorization techniques) return a *NumPy* float array when the *get_embedding* method is called. The row index corresponds to the position of a single graph in the list of graphs inputted. In the same way, columns represent the embedding dimensions.

- *Community detection procedures* return a dictionary when the *get_memberships* method is called. Node indices are keys and the values corresponding to the keys are the community memberships of vertices. Certain graph clustering techniques create a node embedding in order to find vertex clusters. These return a *NumPy* float array when the *get_embedding* method is called. This array is structured like the ones returned by node embedding algorithms.

We demonstrate the standardized output generation and interfacing by the code fragment in Figure 1.4. We create clusterings of a random graph and return dictionaries containing the cluster memberships. Using the external community library we can calculate the modularity of these clusterings (lines 15-16). This shows that the standardized output generation makes interfacing with external graph mining and machine learning libraries easy.

Limitations

The current design of *Karate Club* has certain limitations and we make strong assumptions about the input. We assume that the *NetworkX* graph is undirected and consists of a single strongly connected component. All algorithms assume that nodes are indexed with integers consecutively and the starting node index is 0. Moreover, we assume that the graph is not multipartite, nodes are homogeneous and edges are unweighted (each edge has a unit weight).

In the case of the whole graph embedding algorithms (de Lara and Edouard, 2018; Verma and Zhang, 2017; Narayanan et al., 2017; Chen and Koga, 2019; Tsitsulin et al., 2018; Gao, Wolf and Hirn, 2019) all graphs in the set of graphs must amend the previously listed requirements with respect to the input. The Weisfeiler-Lehman feature-based embedding techniques (Narayanan

DOI: 10.14754/CEU.2024.08

et al., 2017; Chen and Koga, 2019) allow nodes to have a single string feature that can be accessed with the *feature* key. Without the presence of this key, these algorithms default to the use of degree centrality as a node feature.

1.1.5 Experimental Evaluation

In the experimental evaluation of *Karate Club* we will demonstrate two things. First, we will show that the implemented algorithms have a good performance with respect to embedding and extracted community quality on a variety of machine learning problems. Second, we support evidence that those algorithms which in theory scale linearly with the input size (number of nodes or number of graphs) scale linearly using our framework in practice. Throughout these experiments, we will always use the standard hyperparameter settings of the 1.0. release of our package.

Learning performance

The evaluation of the representation quality focuses on three types of machine learning tasks. These are community detection with ground truth communities, node classification with node embeddings, and whole graph classification with graph-level embeddings.

Datasets To evaluate the performance of vertex-level algorithms (node embedding and community detection) we used attributed web, collaboration, and social networks which are publicly available on *SNAP* (Rozemberczki, Allen and Sarkar, 2019; Leskovec and Krevl, 2014). We decided to use attributed networks because a large number of algorithms in *Karate Club* can exploit the presence of node features. These datasets are the following:

- *Wikipedia Crocodiles:* In this graph nodes represent Wikipedia pages and edges are mutual links. The vertex features describe the presence of nouns in the article and the binary target variable indicates the volume of traffic on the site.

DOI: 10.14754/CEU.2024.08

- *GitHub Developers:* Vertices in this network are developers who use GitHub and edges represent mutual follower relationships between the users. Features are derived based on location, biography and other metadata, the binary target variable is whether someone is a machine learning or web developer.

- *Twitch England:* Nodes of this graph are Twitch users from England and edges are mutual friendships between them. Node features were extracted based on the streaming history of the users while the binary node class describes whether the user creates explicit content.

- *Facebook Page-Page:* A network of verified Facebook pages where nodes are pages and the links between nodes are mutual likes. Features are distilled from the page descriptions and the target is the category of the Facebook page (Politicians, Governments, Companies, TV Shows).

The descriptive statistics of these node-level datasets are summarized in Table 1.1. As one can see these networks have a large variety of sizes, levels of clustering, and diameter.

Table 1.1: The social networks used for node level algorithms.

	Wikipedia Crocodiles	**GitHub Developers**	**Twitch England**	**Facebook Page-Page**
Nodes	11,631	37,700	7,126	22,470
Density	0.003	0.001	0.002	0.001
Transitivity	0.026	0.013	0.042	0.232
Diameter	11	7	10	15
Features	13,183	4,005	2,545	4,714

Graph level embedding algorithms were evaluated on a variety of web and social graph datasets which we collected specifically for this paper. We made these graph collections publicly available.[1] The graph collections used for predictive performance evaluation are the following:

- *Reddit Threads:* Discussion and non-discussion based threads from Reddit which we collected in May 2018. The task is to predict whether a thread is discussion-based.

[1]https://snap.stanford.edu/data/

DOI: 10.14754/CEU.2024.08

- *Twitch Egos:* The ego-nets of Twitch users who participated in the partnership program in April 2018. The binary classification task is to predict using the ego-net whether the central gamer plays a single or multiple games.

- *Github Stargazers:* The social networks of developers who starred popular machine learning and web development repositories until 2019 August. The task is to decide whether a social network belongs to a web or machine learning repository.

- *Deezer Egos:* The ego-nets of Eastern European users collected from the music streaming service Deezer in February 2020. The related task is the prediction of gender for the ego node in the graph.

Table 1.2: Statistics of graph datasets used for graph level algorithms.

		Nodes		**Density**		**Diameter**	
Dataset	**Graphs**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Reddit Threads	203,088	11	97	0.021	0.382	2	27
Twitch Egos	127,094	14	52	0.038	0.967	1	2
GitHub StarGazers	12,725	10	957	0.003	0.561	2	18
Deezer Egos	9,629	11	363	0.015	0.909	2	2

We listed the size of these datasets with the respective descriptive statistics in Table 1.2. It is worth noting that the *Reddit Threads* and *Twitch Egos* both have at least 10 fold more graphs than the social graph datasets which are widely used for graph classification evaluation (Yanardag and Vishwanathan, 2015). We would also like to emphasize that the use of graph kernels would not be feasible on graph datasets that are this numerous.

Community Detection We evaluate the community detection performance by running the clustering algorithms on the node level datasets. In case of overlapping community detection algorithms (Yang and Leskovec, 2013; Ye, Chen and Zheng, 2018; Wang et al., 2017; Sun et al., 2017; Kuang, Ding and Park, 2012; Epasto, Lattanzi and Paes Leme, 2017) we assigned each

Table 1.3: Mean NMI values with standard errors on the node level datasets calculated from 100 runs.

	Wikipedia Crocodiles	**GitHub Developers**	**Twitch England**	**Facebook Page-Page**
DANMF Ye, Chen and Zheng (2018)	.051 ± .001	.083 ± .001	.007 ± .001	.164 ± .001
M-NMF Wang et al. (2017)	.063 ± .001	.084 ± .001	.004 ± .001	.068 ± .001
NNSED Sun et al. (2017)	.063 ± .001	.034 ± .001	.004 ± .001	.072 ± .001
SymmNMF Kuang, Ding and Park (2012)	.062 ± .001	.074 ± .001	.007 ± .001	.206 ± .001
Ego-Splitting Epasto, Lattanzi and Paes Leme (2017)	.157 ± .001	**.202 ± .001**	**.223 ± .001**	.346 ± .001
EdMot Li et al. (2019*b*)	.085 ± .001	.180 ± .001	.008 ± .001	.272. ± .001
LabelProp Raghavan, Albert and Kumara (2007)	.119 ± .001	.090 ± .002	.003 ± .001	.320 ± .004
SCD Prat-Pérez, Dominguez-Sal and Larriba-Pey (2014)	**.181 ± .001**	.189 ± .001	.169 ± .001	**.386 ± .001**
GEMSECRozemberczki et al. (2019)	.102 ± .001	.127 ± .001	.008 ± .002	.244 ± .001

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

node to the cluster that has the strongest affiliation score with the node (ties were broken randomly). The metric used for the clustering performance measurement is the average normalized mutual information (henceforth NMI) score calculated between the cluster membership vector and the factual class memberships. We report in Table 1.3 the NMI averages with the standard errors calculated from 100 experimental runs.

Looking at Table 1.3 first we notice that the non-overlapping community detection techniques (Li et al., 2019*b*; Raghavan, Albert and Kumara, 2007; Prat-Pérez, Dominguez-Sal and Larriba-Pey, 2014; Rozemberczki et al., 2019; Epasto, Lattanzi and Paes Leme, 2017) materially outperform the overlapping models which create latent spaces (Yang and Leskovec, 2013; Ye, Chen and Zheng, 2018; Wang et al., 2017; Sun et al., 2017; Kuang, Ding and Park, 2012) on every dataset in terms of NMI. Second, those algorithms that create clusters based on the presence of closed triangles (SCD (Prat-Pérez, Dominguez-Sal and Larriba-Pey, 2014) and Ego-Splitting (Epasto, Lattanzi and Paes Leme, 2017)) have a generally strong performance. Finally, on problems where it can be assumed that the class membership vector is associated with structural properties (e.g. Wikipedia Crocodiles), the overlapping latent space creating community detection methods perform poorly in terms of NMI.

Graph classification In each dataset we created representations for the graphs and used those as predictors for the downstream classification task. We repeated the feature distillation and supervised model training 100 times, using 80% of graphs for training and 20% for testing with seeded splits. Using the graph class vectors of the test set and class probabilities outputted by the logistic regression classifier we calculated the mean area under the curve (henceforth AUC) values which are presented in Table 1.4 along with their standard errors.

The results presented in Table 1.4 show that the representations created by implicit factorization (Narayanan et al., 2017; Chen and Koga, 2019) and spectral finger printing (de Lara and Edouard, 2018; Tsitsulin et al., 2018; Verma and Zhang, 2017; Rozemberczki and Sarkar, 2020) techniques are predictive on most problems. In addition, we see evidence that algorithms

DOI: 10.14754/CEU.2024.08

Table 1.4: Mean AUC values with standard errors on the graph level datasets calculated from 100 seed train-test splits.

	Reddit Threads	**Twitch Egos**	**GitHub StarGazers**	**Deezer Egos**
GL2Vec Chen and Koga (2019)	.753 ± .002	.664 ± .002	.551 ± .001	.504 ± .001
Graph2Vec Narayanan et al. (2017)	.804 ± .002	.702 ± .003	.585 ± .001	.512 ± .001
SF de Lara and Edouard (2018)	.814 ± .002	.678 ± .003	.558 ± .001	.501 ± .001
NetLSD Tsitsulin et al. (2018)	**.827 ± .001**	.631 ± .002	.632 ± .001	.522 ± .001
FGSD Verma and Zhang (2017)	.825 ± .002	.705 ± .003	.656 ± .001	.526 ± .001
GeoScattering Gao, Wolf and Hirn (2019)	.800 ± .001	.697 ± .001	.546 ± .003	.522 ± .003
FEATHER Rozemberczki and Sarkar (2020)	**.830 ± .002**	**.720 ± .003**	**.748 ± .002**	**.540 ± .001**

from the latter group create somewhat higher-quality representations.

Node classification In this series of experiments we evaluated the node classification performance on the node-level datasets. For each graph we learned a node embedding and used the features of this node embedding as predictors for a downstream logistic (softmax) regression model. We repeated the embedding and supervised model training 100 times, using 80% of the nodes for training and 20% for testing with seeded splits. Using the target vectors of the test set and the class probabilities outputted by the downstream model we calculated mean AUC scores. These average AUC values are reported in Table 1.5 with standard errors. The results in Table 1.5 generally demonstrate that the included neighbourhood based (Perozzi, Al-Rfou and Skiena, 2014; Perozzi et al., 2017; Rozemberczki and Sarkar, 2018; Cao, Lu and Xu, 2015; Qiu et al., 2018; Jundong Li, 2019; Sun and Fevotte, 2014; Ou et al., 2016; Belkin and Niyogi, 2002), structural role preserving (Ahmed et al., 2019; Donnat et al., 2018), and attributed (Rozemberczki, Allen and Sarkar, 2019; Yang et al., 2018; Yang and Yang, 2018; Yang et al., 2015; Bandyopadhyay et al., 2018; Zhang et al., 2018) node embedding techniques all generate reasonable quality representations for this classification task. There are additional conclusions; (i) multi-scale node embeddings such as *GraRep* (Cao, Lu and Xu, 2015), *Walklets*, (Perozzi

DOI: 10.14754/CEU.2024.08

et al., 2017), and *MUSAE* (Rozemberczki, Allen and Sarkar, 2019) create high-quality node features, (ii) combining neighbourhood and attribute information results in the best representations (Rozemberczki, Allen and Sarkar, 2019; Zhang et al., 2018), (iii) there is not a single model which is generally superior.

Shapley values of node embedding methods in model ensembles We further evaluate the node embedding methods implemented in Karate Club using Shapley values. Chapter 2 provides an extensive discussion on how the Shapley value, a solution concept from cooperative game theory can be utilized to assign a value to models in an ensemble.

In this experiment, we calculated the Shapley value of each node embedding technique using the same four datasets. We embedded each network 100 times using each technique and used seeded 80%-20% train-test splits to estimate logistic regressions. We then calculated the mean AUC score of each model coalition (altogether 2^18 coalitions) across the 100 samples. We aggregated the forecasts of the logistic regressions across embedding techniques by taking the mean of the forecasted latent probabilities. The Shapley values of the embedding methods for each dataset are presented in Table 1.6.

Contrary to our previous evaluation results, this measure reveals clearly superior embedding methodologies. Attributed methods (Rozemberczki, Allen and Sarkar, 2019; Zhang et al., 2018) consistently receive very high Shapley values across all datasets, meaning that their inclusion in model ensembles results the largest average marginal improvement in AUC scores across all feasible model ensembles.

Scalability We perform scalability tests for all three types of algorithms (community detection, node, and whole graph embedding). For each of these categories, we investigate the scalability of 4 chosen algorithms. We use Erdos-Renyi graphs where the input size and graph density can be manipulated directly.

Figure 1.5 plots runtime against the size and density of the clustered graph while the average

DOI: 10.14754/CEU.2024.08

Table 1.5: Mean AUC values with standard errors on the node level datasets calculated from 100 seed train-test splits.

	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
BoostNE Jundong Li (2019)	.685 ± .001	.845 ± .001	.576 ± .001	.752 ± .001
NodeSketch Yang et al. (2019)	.722 ± .001	.631 ± .001	.520 ± .001	.579 ± .001
Diff2Vec Rozemberczki and Sarkar (2018)	.832 ± .001	.858 ± .001	.589 ± .001	.873 ± .001
NetMF Qiu et al. (2018)	.866 ± .001	.867 ± .001	.629 ± .002	.946 ± .001
Walklets Perozzi et al. (2017)	.875 ± .001	.899 ± .002	.622 ± .001	.973 ± .001
HOPE Ou et al. (2016)	.870 ± .001	.844 ± .001	.612 ± .001	.909 ± .001
GraRep Cao, Lu and Xu (2015)	.888 ± .002	.876 ± .001	.609 ± .001	.952 ± .001
DeepWalk Perozzi, Al-Rfou and Skiena (2014)	.850 ± .001	.872 ± .002	.597 ± .002	.877 ± .001
NMF-ADMM Sun and Fevotte (2014)	.747 ± .001	.784 ± .001	.619 ± .001	.937 ± .001
LAP Belkin and Niyogi (2002)	.784 ± .001	.529 ± .001	.511 ± .001	.501 ± .001
GraphWave Donnat et al. (2018)	.517 ± .001	.620 ± .001	.583 ± .001	.613 ± .001
Role2Vec Ahmed et al. (2019)	.845 ± .001	.862 ± .002	.601 ± .002	.903 ± .002
BANE Yang et al. (2018)	.866 ± .002	.570 ± .001	.551 ± .001	.970 ± .002
TENE Yang and Yang (2018)	.907 ± .001	.874 ± .001	.615 ± .001	.886 ± .001
TADW Yang et al. (2015)	.896 ± .001	.817 ± .001	.612 ± .002	.871 ± .001
FSCNMF Bandyopadhyay et al. (2018)	.912 ± .001	.856 ± .002	.621 ± .001	.891 ± .001
SINE Zhang et al. (2018)	.904 ± .001	**.910 ± .002**	**.646 ± .001**	.979 ± .001
MUSAE Rozemberczki, Allen and Sarkar (2019)	**.931 ± .001**	.903 ± .001	.628 ± .001	**.981 ± .001**

number of edges is fixed to be 10. In the densification scenario, we clustered a graph with 2^{12} nodes. Non-overlapping community detection techniques show a remarkable scalability with respect to the graph size increase, and we also see that the densification of the graph results in longer runtimes.

We measured the same way how the average runtime of node embedding varies with input size changes and densification and plotted these in Figure 1.6. These results show that under no preferential attachment, all of the included methods scale linearly with input size changes. Moreover, implicit factorization runtimes are unaffected by the densification of the graph.

DOI: 10.14754/CEU.2024.08

Table 1.6: Shapley values of embedding methods using the node level datasets calculated from 100 seed train-test splits.

	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
BoostNE Jundong Li (2019)	0.0296	0.0408	0.0439	0.0283
NodeSketch Yang et al. (2019)	0.0443	0.0336	0.0442	0.0314
Diff2Vec Rozemberczki and Sarkar (2018)	0.0459	0.0654	0.0619	0.0496
NetMF Qiu et al. (2018)	0.0594	0.0669	**0.0657**	0.0658
Walklets Perozzi et al. (2017)	0.0572	0.0743	0.0612	**0.0726**
HOPE Ou et al. (2016)	0.0623	0.0608	0.0575	0.0589
GraRep Cao, Lu and Xu (2015)	0.0565	0.0535	0.0566	0.0624
DeepWalk Perozzi, Al-Rfou and Skiena (2014)	0.0579	0.0672	0.0572	0.0571
NMF-ADMM Sun and Fevotte (2014)	0.0370	0.0304	0.0443	0.0289
LAP Belkin and Niyogi (2002)	0.0596	0.0582	0.0551	0.0613
GraphWave Donnat et al. (2018)	0.0287	0.0313	0.0421	0.0291
Role2Vec Ahmed et al. (2019)	0.0573	0.0334	0.0575	0.0642
BANE Yang et al. (2018)	0.0460	0.0335	0.0443	0.0584
TENE Yang and Yang (2018)	0.0709	0.0665	0.0608	0.0675
TADW Yang et al. (2015)	0.0686	0.0625	0.0585	0.0561
FSCNMF Bandyopadhyay et al. (2018)	0.0704	0.0657	0.0636	0.0635
SINE Zhang et al. (2018)	0.0720	**0.0782**	0.0644	0.0725
MUSAE Rozemberczki, Allen and Sarkar (2019)	**0.0763**	0.0776	0.0612	**0.0726**

In case of the whole graph representation we plotted the average runtime as a function of the number of graphs and their size in Figure 1.7. The base graph used for the first plot had 64 nodes and 5 edges per node and for the second plot, we used 2^{10} graphs. First, a takeaway is that the runtime increases linearly with the size of the dataset assuming that the size of the graphs is homogeneous. Second, the spectral fingerprinting techniques (de Lara and Edouard, 2018; Verma and Zhang, 2017) do not scale well when the size of the graphs is increased.

DOI: 10.14754/CEU.2024.08

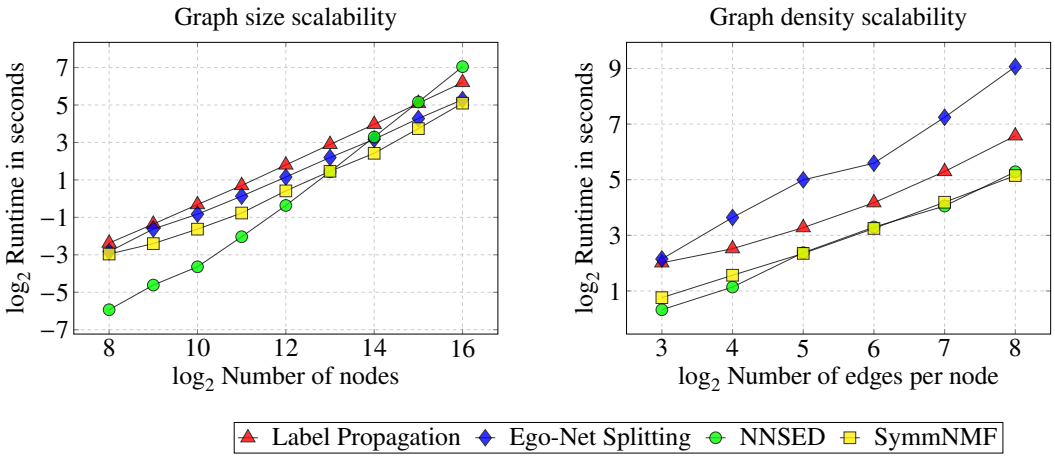


Figure 1.5: Scalability of the community detection procedures in Karate Club. We vary the number of nodes and the density of an Erdos-Renyi graph.

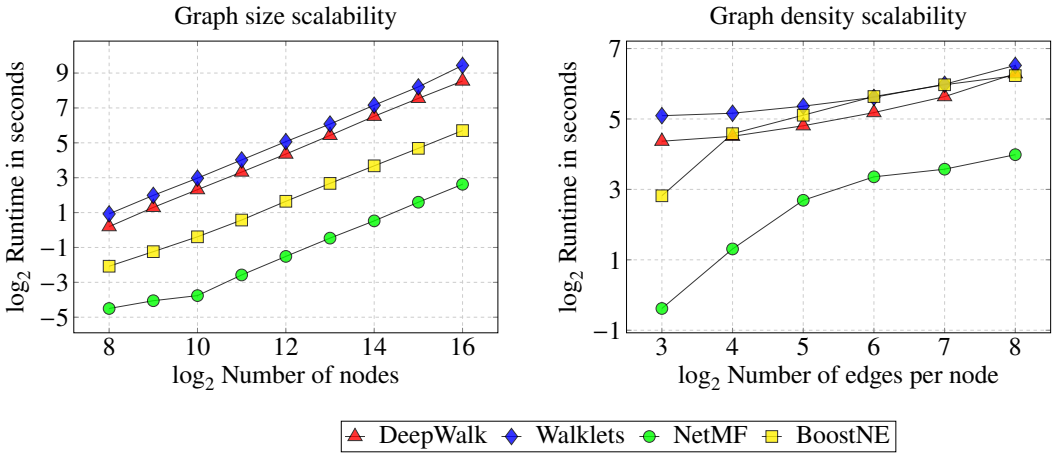


Figure 1.6: Scalability of node embedding procedures in Karate Club. We vary the number of nodes and the density of an Erdos-Renyi graph.

1.1.6 Conclusion and Future Directions

In this work we described *Karate Club* a Python framework built on the open source packages *NetworkX* (Hagberg, Swart and S Chult, 2008), *PyGSP* (Defferrard et al., 2017), *Gensim* (Rehurek and Sojka, 2011), *NumPy* (Walt, Colbert and Varoquaux, 2011), and *SciPy Sparse* (Virtanen et al., 2019) which performs unsupervised learning on graph data. Specifically, it supports community detection, node embedding, and whole graph embedding techniques.

We discussed in detail the design principles that we followed when we created *Karate Club*,

DOI: 10.14754/CEU.2024.08

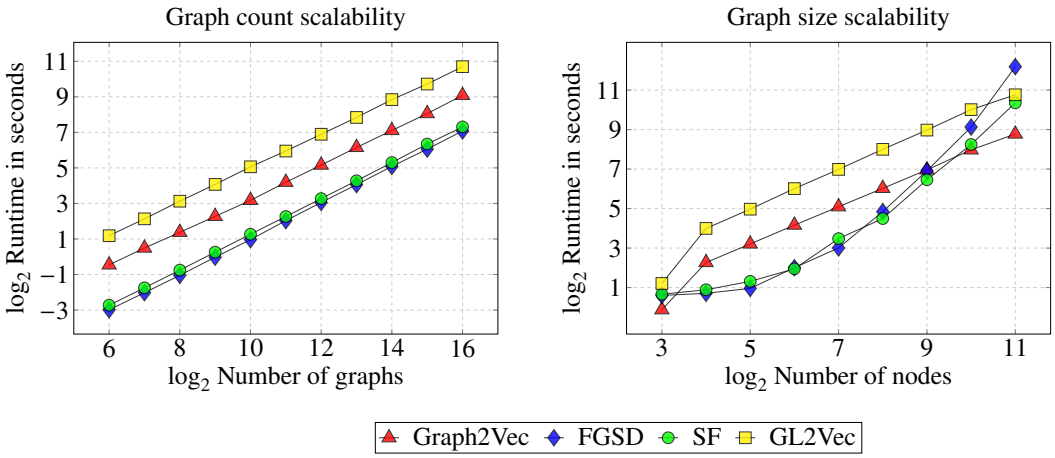


Figure 1.7: Scalability of graph embedding and summarization procedures in Karate Club. We vary the number of Erdos-Renyi graphs and their size.

standard hyperparameter encapsulation, the assumptions about the format of input data and generated output, and available public methods. In order to demonstrate these principles we included illustrative examples of code. In a series of experiments on real-world datasets we validated that the machine learning models in *Karate Club* produce high-quality clusters and embeddings. We also demonstrated on synthetic data that the linear runtime algorithms scale well with increasing input size.

As discussed, *Karate Club* has certain limitations with regard to the types of graphs that it can handle. In the future, we plan to extend it to operate on directed and weighted graphs. Another aim is to provide a general framework for unsupervised learning algorithms on heterogeneous, multiplex, temporal graphs and procedures for the hyperbolic embedding of nodes (Sarkar, 2011; Verbeek and Suri, 2014).

A potential future research application of our library is finding parametrizations of the implemented models that perform well on a wide range of real-life datasets. While for some hyperparameters there is a clear intuition regarding the choice of the proper parameter (e.g.: increasing the number of embedding dimensions above a certain point results in overfitting), the literature lacks evidence on most of the hyperparameters. We present a limited proof of concept on this potential future application in the Appendix of this paper.

DOI: 10.14754/CEU.2024.08

1.1.7 Appendix

We provide a proof of concept using a limited number of methods and parameter space for the future research direction outlined in Section 1.1.6.

In this experiment, we evaluate two methods, HOPE (Ou et al., 2016) and NetMF (Qiu et al., 2018) using four real-life datasets and a limited section of the hyperparameter space. For each hyperparameter constellation, we embed the networks 20 times, and train logistic regressions using seeded 80-20% train-test splits. We evaluate the performance using the mean AUC across the 20 experiments.

In case of HOPE, the parameter space consists only of the embedding dimensions. This experiment should therefore provide insights on how the model performance changes with increasing the number of embedding dimensions. There is a clear intuition regarding this parameter: the higher it is, the more information can be retained by the model (each node is embedded in higher and higher dimensional spaces). At the same time, increasing this parameter may result in overfitting. The results of this experiment are presented in Table 1.7.

The results align with the intuition. We can observe a general positive association between the number of embedding dimensions and the AUC score across all but one datasets. The only exception, Twitch, showcases the overfitting phenomenon. This dataset - being the smallest in terms of the number of nodes among our test cases - benefits from increasing the number of dimensions only until 32 dimensions. Above this level it shows clear signs of overfitting, with the mean AUC scores decreasing both for 64 and 128 dimensions. These results imply that the number of embedding dimensions should depend on the number of data points embedded.

The second experiment uses NetMF (Qiu et al., 2018), which has a 4-dimensional hyperparameter space. We embed each network 20 times and perform our downstream tasks just like it has been done using HOPE. Our evaluation is identical as well, we calculate the mean AUC across the 20 experiments. We use the following subset of hyperparameters for this experiment:

- Embedding dimensions: 8, 16 or 32

DOI: 10.14754/CEU.2024.08

Table 1.7: Mean AUC values with standard errors on the node level datasets calculated from 20 seed train-test splits for different parametrizations of the HOPE method

HOPE Dimensions	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
8	.610 ± .001	.691 ± .001	.574 ± .001	.771 ± .001
16	.735 ± .001	.814 ± .001	.586 ± .001	.779 ± .001
32	.753 ± .001	.834 ± .001	.599 ± .001	.787 ± .001
64	.814 ± .001	.843 ± .001	.594 ± .001	.794 ± .001
128	.876 ± .001	.848 ± .001	.588 ± .001	.908 ± .001

- Number of singular value decomposition (SVD) iterations: 5, 10 or 15
- Pointwise mutual information (PMI) matrix orders: 1 or 2
- Number of negative samples: 1, 2 or 3

Due to the high dimensionality of the hyperparameter space, we focus our analysis on the marginals of the distribution. Table 1.8 evaluates the results for each dataset across different embedding dimensions, Table 1.9 presents the results for each SVD iteration option, Table 1.11 evaluates the optimal choice of PMI order and Table 1.10 contains the results on the number of negative samples.

The results show that the predictive performance is increasing in the embedding dimensions across all datasets. This result is likely to be due to the limitations of this evaluation. It is well known, and is illustrated in the previous experiment that above a certain cutoff, increasing the number of dimensions results in overfitting. Since all evaluations use a relatively low dimension, this pattern is not observed in this experiment.

The number of SVD iterations and negative samples have no significant effect on the mean AUC score for any of the datasets. These results are likely due to the limited number of datasets that have been evaluated in these experiments. Extending the set of real-life networks in this experiment is therefore an important potential future research direction.

In terms of PMI orders, the results are mixed. For two datasets, GitHub and Twitch, choosing

DOI: 10.14754/CEU.2024.08

different PMI orders does result in a significant difference. In case of Wikipedia and Facebook choosing a higher PMI order significantly increases the performance of the downstream estimator in terms of the mean AUC score. These networks are different in terms of their diameters: increasing the PMI order affects networks with a higher diameter.

We have to note that the results above have been established using a limited set of networks and a limited set of hyperparameters, and therefore they only serve the purpose of a proof of concept. Future research is needed to analyze the properties of these embedding methods across additional datasets and segments of the hyperparameter space.

Table 1.8: Mean AUC values with standard errors on the node level datasets calculated from 20 seed train-test splits for different parametrizations of the NetMF method across embedding dimensions

NetMF Dimensions	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
8	.725 ± .041	.856 ± .003	.613 ± .008	.772 ± .024
16	.810 ± .040	.864 ± .001	.613 ± .003	.826 ± .037
32	.844 ± .024	.871 ± .002	.618 ± .004	.889 ± .059

Table 1.9: Mean AUC values with standard errors on the node level datasets calculated from 20 seed train-test splits for different parametrizations of the NetMF method across SVD iterations

NetMF SVD iterations	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
5	.791 ± .006	.864 ± .006	.614 ± .006	.828 ± .007
10	.793 ± .006	.864 ± .006	.615 ± .006	.829 ± .006
15	.794 ± .006	.864 ± .006	.615 ± .006	.831 ± .006

DOI: 10.14754/CEU.2024.08

Table 1.10: Mean AUC values with standard errors on the node level datasets calculated from 20 seed train-test splits for different parametrizations of the NetMF method across negative sample numbers

NetMF Negative samples	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
1	.794 ± .006	.863 ± .006	.615 ± .007	.831 ± .007
2	.795 ± .006	.865 ± .006	.616 ± .002	.826 ± .006
3	.788 ± .006	.864 ± .008	.612 ± .003	.832 ± .006

Table 1.11: Mean AUC values with standard errors on the node level datasets calculated from 20 seed train-test splits for different parametrizations of the NetMF method across PMI orders

NetMF PMI orders	Wikipedia Crocodiles	GitHub Developers	Twitch England	Facebook Page-Page
1	.766 ± .051	.864 ± .006	.616 ± .003	.806 ± .021
2	.829 ± .059	.864 ± .007	.613 ± .007	.854 ± .081

DOI: 10.14754/CEU.2024.08

1.2 Little Ball of Fur: A Python Library for Graph Sampling

1.2.1 Introduction

Modern graph datasets such as social networks and web graphs are large and can be mined to extract detailed insights. However, the large size of the datasets poses fundamental computational challenges on graphs (Kang, Tsourakakis and Faloutsos, 2009; Gonzalez et al., 2012). Exploratory data analysis and computation of basic descriptive statistics can be time-consuming on real-world graphs. More advanced graph mining techniques such as node and edge classification or clustering can be completely intractable on full-size datasets such as web graphs.

One of the fundamental techniques to deal with large datasets is sampling. On simple datasets such as point clouds, sampling preserves most of the distributional features of the data and forms the basis of machine learning (Shalev-Shwartz and Ben-David, 2014). However, graphs represent complex interrelations, so that naive sampling can destroy the salient features that constitute the value of the graph data. Graph sampling algorithms therefore need to be sensitive to the various features that are relevant to the downstream tasks. Such features include statistics such as diameter, clustering coefficient (Easley, Kleinberg et al., 2010), transitivity or degree distribution. In more complex situations, graphs are used for community detection, classification, or edge prediction (Hamilton, Ying and Leskovec, 2017). A sampling algorithm should be representative with respect to such downstream learning tasks.

Extracting sets of vertices and edges that result in a representative subsample of the original source graph is a nontrivial problem to solve. The complicated nature of sampling is evident as one has to address multiple specific questions about how sampling affects certain aspects of the follow-up data analysis. What is the optimal ratio of nodes and edges retained after the sampling? How does sampling affect the connectivity of the sampled graph compared to that of the source graph? Does sampling change the value of graph-level descriptive statistics of structure such as diameter, average degree, transitivity, or the degree correlation? How distributional characteristics of the node and edge level graph are changing due to the sampling?

DOI: 10.14754/CEU.2024.08

How competitive are the post graph sampling trained embeddings, classifiers, and clustering models? What is the nature of the trade-off between the scalability gains and the downstream analysis quality?

Various graph sampling procedures have been proposed with different objectives (Hu and Lau, 2013). The implementation of the graph sampling technique and the choice of parameters used for the subgraph extraction can affect its utility for the task in question. A toolbox of well-understood graph sampling techniques can make it easier for researchers and practitioners to easily perform graph sampling and have consistent reproducible sampling across projects. Our goal is to make a large number of graph sampling techniques available to a large audience.

We release *Little Ball of Fur* – an open-source Python library for graph sampling. This is the first publicly available and documented Python graph sampling library. The general design of our framework is centered around an end-user-friendly application public interface which allows for fast-paced development and interfacing with other graph mining frameworks.

We achieve this by applying a few core software design principles consistently. Sampling techniques are implemented as individual classes and have pre-parametrized constructors with sensible default settings, which include the number of sampled vertices or edges, a random seed value, and hyperparameters of the sampling method itself. Algorithmic details of the sampling procedures are implemented as private methods to shield the end-user from the inner mechanics of the algorithm. These concealed workings of samplers rely on the standard Python library and Numpy (Walt, Colbert and Varoquaux, 2011). Practically, sampling techniques only provide a single shared public method (sample) which returns the sampled graph. Sampling procedures use NetworkX (Hagberg, Swart and S Chult, 2008) and NetworKit (Staudt, Sazonovs and Meyerhenke, 2016) graphs as the input and the output adheres to the same widely used generic formats.

We demonstrate the practical applicability of our framework on various social networks and web graphs (e.g. Facebook, LastFM, Deezer, Wikipedia). We show that our package allows the precise estimation of macro-level statistics such as average degree, transitivity, and degree

DOI: 10.14754/CEU.2024.08

correlation. We provide evidence that the use of sampling routines from *Little Ball of Fur* can reduce the runtime of node and whole graph embedding algorithms. Using these embeddings as input features for node and graph classification tasks we establish that the embeddings learned on the subsampled graphs extract high-quality features.

The rest of this paper has the following structure. In Section 1.2.2 we overview the relevant literature about graph sampling. This discussion covers node, edge, and exploration sampling techniques, and the possible applications of sampling from graphs. The design principles that we followed when *Little Ball of Fur* was developed are discussed in Section 1.2.3 with samples of illustrative Python code. The subsampling techniques provided by our framework are evaluated in Section 1.2.4. We present results about network statistic estimation performance, and machine learning case studies about node and graph classification. The paper concludes with Section 1.2.5 where we discuss our main findings and point out directions for future work. We open-sourced the software package and it can be downloaded from `https://github.com/benedekrozemberczki/littleballoffur`; the Python package can be installed via the *Python Package Index*. A comprehensive documentation can be accessed at `https://little-ball-of-fur.readthedocs.io/en/latest/` with a step-by-step tutorial.

1.2.2 Related work

In this section we briefly overview the types of graph subsampling techniques included in *Little Ball of Fur* and the node and graph level representation learning algorithms used for the experimental evaluation of the framework.

Graph sampling techniques

Graph subsampling procedures have three main groups – node, edge, and exploration-based techniques. We give a brief overview of these techniques in this section.

DOI: 10.14754/CEU.2024.08

Node sampling Techniques which sample nodes select a set of representative vertices and extract the induced subgraph among the chosen vertices. Nodes can be sampled uniformly without replacement (RN) (Stumpf, Wiuf and May, 2005), proportional to the degree centrality of nodes (RDN) (Adamic et al., 2001) or according to the pre-calculated PageRank score of the vertices (PRN) (Leskovec and Faloutsos, 2006). All of these methods assume that the set of vertices and edges in the graph is known ex-ante.

Edge sampling The simplest link sampling algorithm retains a randomly selected subset of edges by sampling those uniformly without replacement (RE) while another approach is to randomly select nodes and an edge that belongs to the chosen node (RNE) (Krishnamurthy et al., 2005). These techniques can be hybridized by alternating between node-edge sampling and random edge selection with a parametrized probability (HRNE) (Krishnamurthy et al., 2005).

By randomly selecting a set of retained edges one implicitly samples nodes. Because of this, the random edge selection can be followed up by an induction step (TIES) (Ahmed, Neville and Kompella, 2013) in which the additional edges among chosen nodes are all added. This step can be a partial induction (PIES) (Ahmed, Neville and Kompella, 2013) if the edges were sampled in a streaming fashion and only edges with already sampled nodes are selected in the induction step.

Exploration-based sampling Node and edge sampling techniques do not extract representative subsamples of a graph by exploring the neighbourhoods of seed nodes. In comparison exploration-based sampling techniques probe the neighborhood of a seed node or a set of seed vertices.

A group of exploration-based sampling techniques uses search strategies on the graph to extract a subsample. The simplest search-based strategies include classical traversal methods such as breadth-first search (BFS) and depth-first search (DFS) (Doerr and Blenn, 2013).

DOI: 10.14754/CEU.2024.08

Snowball sampling (SB) (Goodman, 1961) is a restricted version of BFS where a maximum fixed k number of neighbors is traversed. Forest fire (FF) sampling (Leskovec, Kleinberg and Faloutsos, 2005) is a parametrized stochastic version of SB sampling where the constraint on the maximum number of traversed neighbours only holds in expectation. A local greedy search-based technique is community structure expansion sampling (Maiya and Berger-Wolf, 2010) (CSE) which starting with a seeding node adds new nodes to the sampled set which can reach the largest number of unknown nodes. Another simpler search-based sampling technique is the random node-neighbor (RNN) (Leskovec and Faloutsos, 2006) algorithm which randomly selects a set of seed nodes, takes the neighbors in a single hop, and induces the edges of the resulting vertex set. Searching for shortest paths (SP) (Rezvanian and Meybodi, 2015) between randomly sampled nodes can be used for selecting sequences of nodes and edges to induce a subsampled graph.

A large family of exploration-based graph sampling strategies is based on random walks (RW) (Gjoka et al., 2010). These techniques initiate a random walker on a seed node which traverses the graph and induces a subgraph which is used as the sample. Random walk-based sampling has numerous shortcomings and a large number of sampling methods try to correct for specific limitations.

One of the major limitations is that random walks are inherently biased towards visiting high-degree nodes in the graph (Hu and Lau, 2013), Metropolis-Hastings random walk (MHRW) (Hübler et al., 2008; Stutzbach et al., 2008) and its rejection-constrained variant (RC-MHRW) (Li et al., 2015) address this bias by making the walker prone to visit lower degree nodes.

Another major shortcoming of random walk-based sampling is that the walker might get stuck in the closely-knit community of the seed node. There are multiple ways to overcome this. The first one is the use of non-backtracking random walks (NBTRW) (Lee, Xu and Eun, 2012) which removes the tottering behavior of random walks. The second one is circulating the neighbors of every node with a vertex-specific queue (CNRW) (Zhou, Zhang and Das, 2015). A third strategy involves teleports - the random walker jumps (RWJ) (Ribeiro and Towsley,

DOI: 10.14754/CEU.2024.08

2010) with a fixed probability to a random node from the current vertex. A fourth approach is to make the walker biased towards weak links by creating a common neighbor-aware random walk sampler (CNARW) (Li et al., 2019*c*) which is biased towards neighbors with low neighborhood overlap. A fifth strategy is using multiple random walkers simultaneously which form a so-called frontier of random walkers (FRW) (Ribeiro and Towsley, 2010). These techniques can be combined with each other in a modular way to overcome the limitations of random walk-based sampling.

There are other possible modifications to traditional random walks which we implemented in *Little Ball of Fur*. One example is random walk with restart (RWR) (Leskovec and Faloutsos, 2006), which is similar to RWJ sampling, but the teleport always ends with the seed node or loop erased random walks (LERW) (Wilson, 1996) which can sample spanning trees from a source graph uniformly.

Node and whole graph embedding

Our experimental evaluation includes node and graph classification for which we use features extracted with neighbourhood-preserving node embeddings and whole graph embedding techniques. These experiments utilize the implementations of the embedding techniques implemented in the Karate Club library discussed in Section 1.1.

Neighbourhood preserving node embedding Given a graph $G = (V, E)$ neighbourhood preserving node embedding techniques (Perozzi, Al-Rfou and Skiena, 2014; Tang et al., 2015; Grover and Leskovec, 2016; Ou et al., 2016; Cao, Lu and Xu, 2015; Perozzi et al., 2017; Rozemberczki and Sarkar, 2018) learn a function $f : V \rightarrow \mathbb{R}^d$ which maps the nodes $v \in V$ into a d dimensional Euclidean space. In this embedding space, a pre-determined notion of node-node proximity is approximately preserved by learning the mapping. The vector representations created by the embedding procedure can be used as input features for node classifiers.

DOI: 10.14754/CEU.2024.08

Whole graph embedding and statistical fingerprinting Starting with a set of graphs $\mathcal{G} = (G_1, \ldots, G_n)$ whole graph embedding and statistical fingerprinting procedures (Narayanan et al., 2017; Chen and Koga, 2019; Verma and Zhang, 2017; Tsitsulin et al., 2018; Rozemberczki and Sarkar, 2020) learn a function $h : \mathcal{G} \rightarrow \mathbb{R}^d$ which maps the graphs $G \in \mathcal{G}$ to a d dimensional Euclidean space. In this space, those graphs which have similar structural patterns are close to each other. The vector representations distilled by these whole graph embedding techniques are useful inputs for graph classification algorithms.

1.2.3 Design principles

We overview the core design principles that we applied when we designed *Little Ball of Fur*. Each design principle is discussed with illustrative examples of Python code which we explain in detail.

Encapsulated sampler hyperparameters, random seeding, and parameter inspection

Graph sampling methods in *Little Ball of Fur* are implemented as individual classes which all inherit from the *Sampler* class. A *Sampler* object is created by using the constructor which has default out-of-the-box hyperparameter settings. These default settings are available in the documentation and can be customized by re-parametrizing the *Sampler* constructor. The hyperparameters of the sampling techniques are stored as *public attributes* of the *Sampler* instance which allows for inspection of the hyperparameter settings by the user. Each graph sampling procedure has a seed parameter – this value is used to set a random seed for the standard Python and NumPy random number generators. This way the subsample extracted from a specific graph with a fixed seed is always going to be the same.

The code snippet in Figure 1.8 illustrates the encapsulated hyperparameter and inspection features of the framework. We start the script by importing a simple random walk sampler from the package (line 1). We initialize the first random walk sampler instance without changing the

default hyperparameter settings (line 3). As the seed and hyperparameters are exposed we can print the seed parameter which is a public attribute of the sampler (line 4) and we can see the default value of the seed. We create a new instance with a parametrized constructor which sets a new seed (line 6) that modifies the value of the publicly available random seed (line 7).

```
1 from littleballoffur import RandomWalkSampler
2
3 sampler = RandomWalkSampler()
4 print(sampler.seed)
5
6 sampler = RandomWalkSampler(seed=41)
7 print(sampler.seed)
```

Figure 1.8: Re-parametrizing and initializing the constructor of a random walk sampler by changing the random seed.

Achieving API consistency and non-proliferation of classes

The graph sampling procedures included in *Little Ball of Fur* are implemented with a consistent application public interface. As we discussed the parametrized constructor is used to create the sampler instance and the samplers all have a single available *public method*. The subsample of the graph is extracted by the use of the *sample* method which takes the source graph and calls the private methods of the sampling algorithm.

We limited the number of classes and methods in *Little Ball of Fur* with a straightforward design strategy. First, the graph sampling procedures do not rely on custom data structures to represent the input and output graphs. Second, inheritance from the *Sampler* ensures that private methods that check the input format requirements do not have to be re-implemented on a case-by-case basis for each sampling procedure.

In Figure 1.9, first we import *NetworkX* and the random walk sampler from *Little Ball of Fur* (lines 1-2). Using these libraries we create a Watts-Strogatz graph (line 4) and a random walk sampler with the default hyperparameter settings of the sampling procedure (line 6). We

DOI: 10.14754/CEU.2024.08

```
1 import networkx as nx
2 from littleballoffur import RandomWalkSampler
3
4 graph = nx.watts_strogatz_graph(1000, 10, 0)
5
6 sampler = RandomWalkSampler()
7 sampled_graph = sampler.sample(graph)
8
9 print(nx.transitivity(sampled_graph))
```

Figure 1.9: Using a random walk sampler on a Watts-Strogatz graph without changing the default sampler settings.

sample a subgraph with the public *sample* method of the random walk sampler (line 7) and print the transitivity calculated for the sampled subgraph (line 8).

```
1 import networkx as nx
2 from littleballoffur import ForestFireSampler
3
4 graph = nx.watts_strogatz_graph(1000, 10, 0)
5
6 sampler = ForestFireSampler()
7 sampled_graph = sampler.sample(graph)
8
9 print(nx.transitivity(sampled_graph))
```

Figure 1.10: Using a forest fire sampler on a Watts-Strogatz graph without changing the default sampler settings.

The piece of code presented in Figure 1.9 can be altered seamlessly to perform Forest Fire sampling by modifying the sampler import (line 2) and changing the constructor (line 7) – these modifications result in the example in Figure 1.10.

These illustrative sampling pipelines presented in Figures 1.9 and 1.10 demonstrate the advantage of maintaining API consistency for the samplers. Changing the graph sampling technique that we used only required minimal modifications to the code. First, we replaced the import of the sampling technique from the *Little Ball of Fur* framework. Second, we used the constructor of the newly chosen sampling technique to create a sampler instance. Finally, we

DOI: 10.14754/CEU.2024.08

were able to use the shared *sample* method and the same pipeline to calculate the transitivity of the sampled graph.

Backend deployment, standardized dataset ingestion and limitations

Little Ball of Fur was implemented with a backend wrapper. Sampling procedures can be executed by the *NetworKit* (Staudt, Sazonovs and Meyerhenke, 2016) or *NetworkX* (Hagberg, Swart and S Chult, 2008) backend libraries depending on the format of the input graph. Basic graph operations such as random neighbor or shortest path retrieval of the backend libraries have standardized naming conventions, data generation and ingestion methods. The generic backed wrapper based design allows for the future inclusion of other general graph manipulation backend libraries such as *GraphTool* (Peixoto, 2014) or *SNAP* (Leskovec and Krevl, 2014).

The shared public *sample* method of the node, edge, and exploration-based sampling algorithms takes a *NetworkX*/*Networkit* graph as input and the returned subsample is also a *NetworkX*/*Networkit* graph. The subsampling does not change the indexing of the vertices.

The rich ecosystem of graph subsampling methods and the consistent API required that *Little Ball of Fur* was designed with a limited scope and we made restrictive assumptions about the input data used for sampling. Specifically, we assume that vertices are indexed numerically, the first index is 0 and indexing is consecutive. We assume that the graph that is passed to the sampling method is undirected and unweighted (edges have unit weights). In addition, we assume that the graph forms a single strongly connected component and orphaned nodes are not present. Heterogeneous, multiplex, multipartite, and attributed graphs are not handled by the 1.0 release of the sampling framework.

The sampler classes all inherit private methods that check whether the input graph violates the listed assumptions. These are called within the *sample* method before the sampling process itself starts. When any of the assumptions is violated an error message is displayed for the end-user about the wrong input and the sampling is halted.

DOI: 10.14754/CEU.2024.08

1.2.4 Experimental Evaluation

To evaluate the sampling algorithms implemented in *Little Ball of Fur* we perform a number of experiments on real-world social networks and webgraphs. Details about these datasets are discussed in this subsection. We proceed by showing how randomized spanning tree sampling can be used to speed up node embedding without reducing predictive performance. Our ablation study about graph classification demonstrates how connected graph subsampling can accelerate the application of whole graph embedding techniques. We conclude this subsection by presenting results about estimating graph-level descriptive statistics.

Table 1.12: Statistics of social networks used for comparing sampling and node classification algorithms.

	Facebook Page-Page	**Wikipedia Crocodiles**	**LastFM Asia**	**Deezer Hungary**
Nodes	22,470	11,631	7,624	47,538
Density	0.0007	0.0025	0.0010	0.0002
Transitivity	0.2323	0.0261	0.1786	0.0929
Diameter	15	11	15	12
Labels	4	2	18	84

Datasets

We use real-world social network and webgraph datasets to compare the performance of sampling procedures and test their utility for speeding up classification tasks.

Node level datasets The datasets used for graph statistic estimation and node classification are all available on SNAP (Leskovec and Krevl, 2014), and descriptive statistics can be found in Table 1.12.

- **Facebook Page-Page** (Rozemberczki, Allen and Sarkar, 2019) is a webgraph of verified official Facebook pages. Nodes are pages representing politicians, governmental organizations, television shows, and companies while the edges are links between the pages. The related task is multinomial node classification for the four page categories.

DOI: 10.14754/CEU.2024.08

- **Wikipedia Crocodiles** (Rozemberczki, Allen and Sarkar, 2019) is a webgraph of Wikipedia pages about crocodiles. Nodes are the pages and edges are mutual links between the pages. The potential task is binary node classification.

- **LastFM Asia** (Rozemberczki and Sarkar, 2020) is a social network of LastFM (English music streaming service) users who are located in Asian countries. Nodes are users and links are mutual follower relationships. The task on this dataset is multinomial node classification – one has to predict the location of the users.

- **Deezer Hungary** (Rozemberczki et al., 2019) is a social network of Hungarian Deezer (French music streaming service) users. Nodes are users located in Hungary and edges are friendships. The relevant task is multi-label multinomial node classification - one has to list the music genres liked by the users.

Graph level datasets Our classification study on subsampled sets of graphs utilized forum threads and small-sized social networks of developers. The respective descriptive statistics of these datasets are in Table 1.13.

- **Reddit Threads 10K** (Rozemberczki, Kiss and Sarkar, 2020*a*) is a random subsample of 10 thousand graphs from the original Reddit threads datasets. Threads can be discussion and non-discussion based and the task is the binary classification of them according to these two categories.

- **GitHub StarGazers** (Rozemberczki, Kiss and Sarkar, 2020*a*) is a set of small sized social networks. Each social network is a community of developers who starred a specific machine learning or web development package on Github. The task is to predict the type of the repository based on the community graph.

DOI: 10.14754/CEU.2024.08

Table 1.13: Descriptive statistics and size of the graph datasets for graph subsampling and whole graph classification.

		Nodes		Density		Diameter	
Dataset	**Graphs**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Reddit Threads 10K	10,000	11	97	0.021	0.291	2	22
GitHub StarGazers	12,725	10	957	0.003	0.561	2	18

Node classification with randomly sampled spanning tree embeddings of networks

Node embedding vectors (Perozzi, Al-Rfou and Skiena, 2014; Perozzi et al., 2017) are useful compact descriptors of vertex neighborhoods when it comes to solving classification problems. In traditional classification scenarios, the whole graph is used to learn the node embedding vectors. At the same time, the size of social networks and web graphs can significantly limit the applicability of these methods for researchers with limited computational resources. In this experiment, we study a situation where the embedding vectors are learned from a randomly sampled spanning tree of the original graph. We compare the predictive value of node embeddings learned on the whole graph with ones learned from spanning trees extracted with randomized BFS (Krishnamurthy et al., 2005), DFS (Krishnamurthy et al., 2005) and LERW (Wilson, 1996). The main advantage of randomized spanning trees is that storing the whole graph requires $O(|E|)$ memory when we learn the node embedding. In contrast, storing a sampled spanning tree only requires $O(|V|)$ space.

Experimental settings The experimental pipeline that we used for node classification has four stages.

1. *Graph sampling.* The BFS, DFS and LERW sample-based embeddings start with the extraction of a random spanning tree using *Little Ball of Fur.* This sample is fed to the embedding procedure.

2. *Upstream model.* The sampled graph is fed to the unsupervised upstream models DeepWalk (Perozzi, Al-Rfou and Skiena, 2014) and Walklets (Perozzi et al., 2017) which

DOI: 10.14754/CEU.2024.08

learn the neighborhood preserving node embedding vectors. We used the Karate Club (Rozemberczki, Kiss and Sarkar, 2020*a*) implementation of these models with the default hyperparameter settings.

3. *Downstream model.* We inputted the node embedding vectors as input features for a logistic regression classifier – we used the scikit-learn implementation (Pedregosa et al., 2011) with the default hyperparameter settings. The downstream models were trained with a varying ratio of nodes.

4. *Evaluation.* We report average AUC values on the test set calculated from 100 seeded sampling, embedding and downstream model training runs.

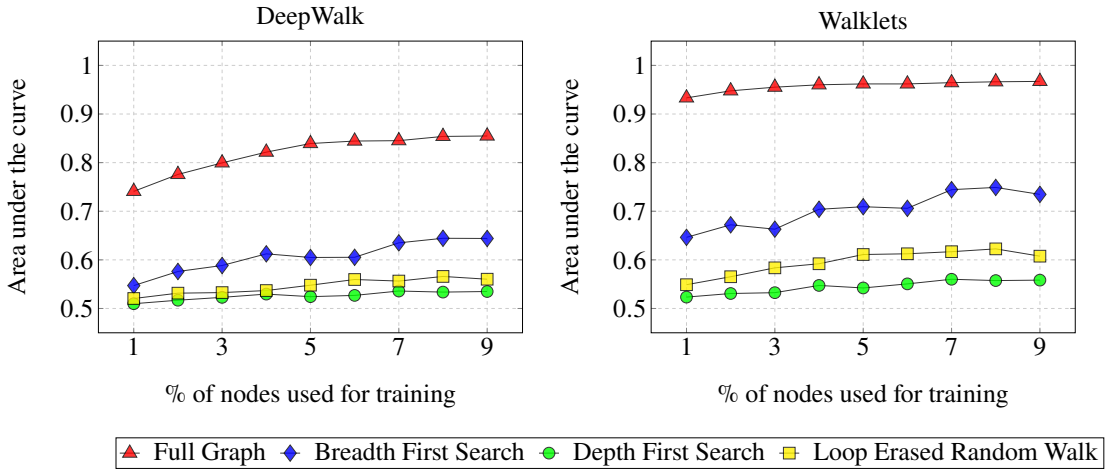


Figure 1.11: Node classification performance on the Facebook Page-Page graph (Rozemberczki, Allen and Sarkar, 2019) evaluated by average AUC scores on the test set calculated from a 100 seeded experimental runs.

Experimental results We report the predictive performance for the Facebook Page-Page and LastFM Asia graphs respectively on Figures 1.11 and 1.12. First, we see that the features extracted from the BFS, DFS and LERW sampled spanning trees are less valuable for node classification based on the predictive performance on these two social networks. In plain words node embeddings of randomly sampled spanning trees produce inferior features. Second, the marginal gains of additional training data are smaller when the embedding is learned from a

DOI: 10.14754/CEU.2024.08

subsampled graph. Third, DFS sampled node embedding features have a considerably lower quality compared to the BFS and LERW sampled node embedding features. Finally, the Walklets based higher order proximity preserving embeddings have a superior predictive performance compared to the DeepWalk based ones even when the graph being embedded is a randomly sampled spanning tree of the source graph. This experiment shows that randomly sampled spanning tree embeddings (especially if trees are extracted using BFS) are valid alternatives to embedding the whole graph in environments with limited computational capacities.

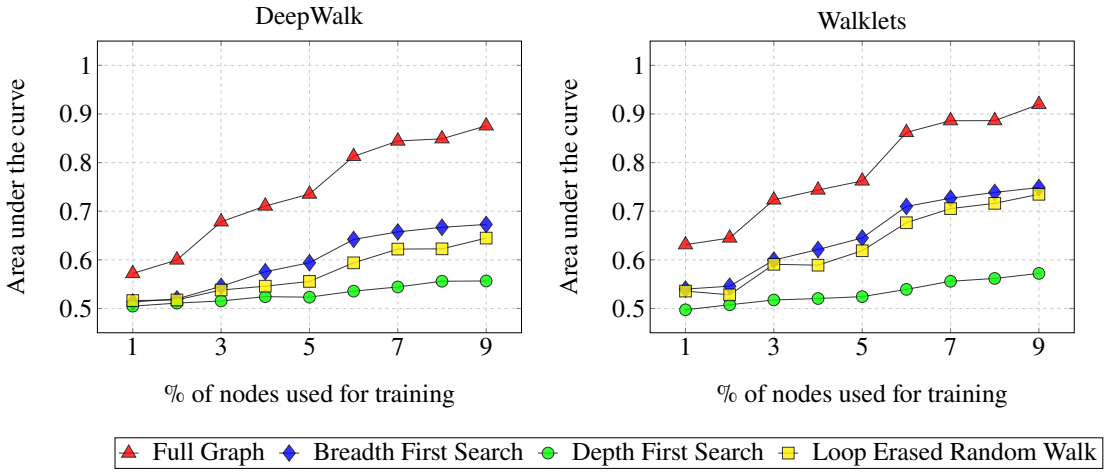
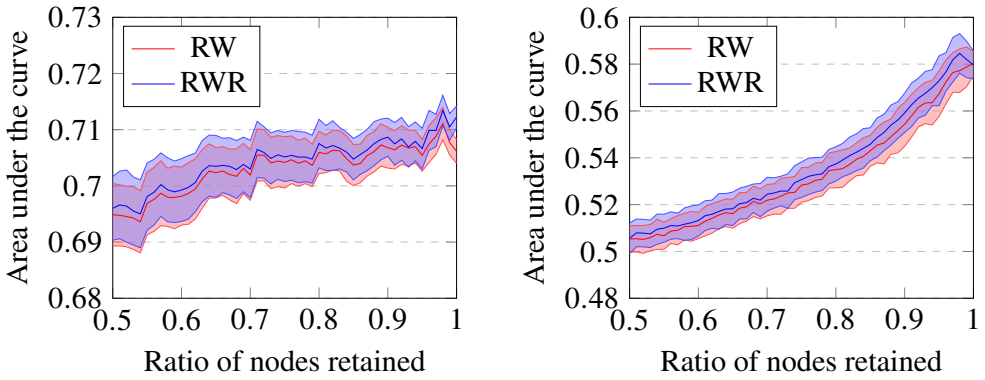


Figure 1.12: Node classification performance on the LastFM Asia graph (Rozemberczki and Sarkar, 2020) evaluated by average AUC scores on the test set calculated from a 100 seeded experimental runs.

An ablation study of graph classification

Graph classification procedures use the whole graph embedding vectors as input to discriminate between graphs based on structural patterns. Using subsamples of the graphs and extracting structural patterns from those can speed up this classification process. We will investigate how exploration-based sampling techniques perform when they are used to obtain the samples used for the embedding. A finding of low performance degradation would imply significant potential future applications. Graph embeddings are computationally expensive, and thus sampling might provide a way to reduce resource requirements and runtime.

DOI: 10.14754/CEU.2024.08



(a) Reddit Threads

(b) Github Stargazers

Figure 1.13: Graph classification performance on the Reddit Threads and GitHub Stargazers graph datasets (Rozemberczki, Kiss and Sarkar, 2020*a*) evaluated by average AUC scores on the test set calculated from 100 seeded experimental runs. We also report standard deviations around the mean performance.

Experimental settings. The data processing that we used for the evaluation of graph classification performance has four stages.

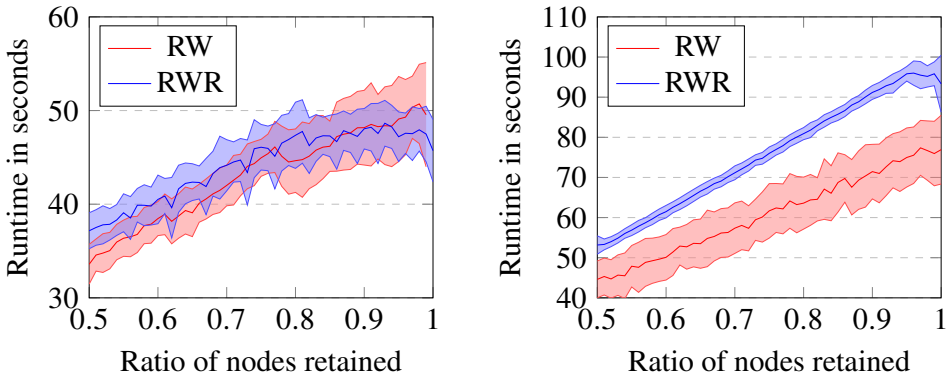
1. *Graph sampling.* We sample both datasets using the RW (Gjoka et al., 2010) and RWR (Leskovec and Faloutsos, 2006) methods implemented in *Little Ball of Fur* 100 times for each retainment rate with different random seeds. Using these algorithms ensures that the graphs' connectivity patterns are unchanged.

2. *Upstream model.* Following the sampling, all of the samples are embedded using the Graph2Vec (Narayanan et al., 2017) algorithm. This procedure uses the presence of subtrees as structural patterns.

3. *Downstream model.* With the embedding vectors as covariates, we estimate a logistic regression for each dataset and retainment rate. We rely on the scikit-learn implementation (Pedregosa et al., 2011) of the classifier with the default hyperparameter settings. We use 80% of the graphs to train the classifier.

4. *Evaluation.* The classification performance is evaluated using the AUC based on the remaining 20% of graphs which form the test set.

DOI: 10.14754/CEU.2024.08



(a) Reddit Threads (b) Github Stargazers

Figure 1.14: Graph embedding runtime on the Reddit Threads and GitHub Stargazers graph datasets (Rozemberczki, Kiss and Sarkar, 2020*a*) calculated from 100 experimental runs. We also report standard deviations around the mean performance.

Experimental results. We report mean AUC values along with a standard deviation band in Figure 1.13 for the Reddit Threads and Github Stargazers datasets with the Random Walk and Random Walk with Restart sampling methods. Lower retainment rate is associated with a lower classification performance, as it can be expected. The more ragged, step function-like pattern observed in case of the Reddit threads dataset is likely to be due to the interplay of structural pattern downsampling and the generally smaller graphs in the dataset.

We report the mean runtime of the graph embedding process with a band of standard deviations in Figure 1.14. As we decrease the retainment rate, a significant decrease in runtime is prevalent. There is a clear trade-off between runtime and predictive performance. It is, however, worth noting that while the runtime associated with the 50% retainment rate is, in most cases close to half of the runtime using the whole graphs, the loss in classification power in the case of the Reddit Threads dataset is less significant.

Estimating descriptive statistics

A traditional task for the evaluation of graph sampling algorithms is the estimation of graph-level descriptive statistics. The graph level descriptive statistic is calculated for the sampled

DOI: 10.14754/CEU.2024.08

graph and it is compared to the ground truth value which is calculated based on the whole set of nodes and edges. A well-performing sampling procedure is ought to give a precise estimate for the graph level quantity of interest with a reasonably small subsample of the graph. Creating a sample that is representative in terms of different network characteristics might be important for many disciplines. Evaluation of these techniques on real-life networks is therefore a crucial experiment.

Experimental settings The pipeline used for estimating the graph-level descriptive statistics had two stages.

1. *Graph sampling.* Node and exploration-based sampling procedures sample 50% of vertices, while the edge sampling techniques select 50% of the edges to extract a subgraph.

2. *Descriptive statistic estimation.* We calculated the average of the clustering coefficient (transitivity), average node degree, and the degree correlation for the sampled graphs. We did 10 seeded experimental runs to get an estimate of the statistics and calculated the standard error around these averages.

Experimental results The ground truth and estimated descriptive statistics are enclosed in Table 1.14 for all of the node-level datasets. Blocks of rows correspond to node, edge, random walk-based, and non-random walk-based exploration sampling algorithms. In each block of methods, bold numbers denote the best-performing sampling technique (closest to the ground truth) in a given category for a specific estimated descriptive statistic and dataset.

We can make a few generic observations about the quality of descriptive statistic estimates. First, there is not a clearly superior sampling technique. This holds generally and within all of the main categories of considered algorithms. Specifically, there is not a superior node, edge, or expansion-based sampling procedure. Second, the induction-based edge sampling techniques (TIES and PIES) give a good estimate of the statistics, but the induction step includes more than 50% of the edges. Because of this, the obtained good estimation performance is somewhat

DOI: 10.14754/CEU.2024.08

misleading as the majority of edges are still retained after the induction step. Third, edge sampling algorithms sometimes fail to estimate the direction of the degree correlation properly. Finally, the random walk-based techniques generally tend to overestimate the average degree. This is not surprising considering that these are biased toward high-degree nodes.

In terms of different ground truth measures, we can make some additional observations. Considering node sampling techniques, the clustering coefficient is best retained by the original random node sampling for all datasets. This method is clearly superior to any other sampling techniques from any other classes in two of our experimental samples, and it consistently provides good quality estimates of the ground truth statistic. Therefore, in research settings that require a representative sample in terms of the clustering coefficient, our results indicate that the best sampling method is random node sampling.

In case of the average degree, there is no single best algorithm even within classes of sampling methods. It is worth noting though, that there are two methods that provide consistently close-to-ground-truth estimates even if they are outperformed in some of the samples. These are the PageRank-weighted random node sampling and the random edge selection method with partial induction (PIES).

In terms of degree correlation, techniques based on random walk tend to perform best. Among these methods, the one using a frontier of random walkers (FRW) and the common neighbor-aware random walk sampler (CNARW) tend to provide decent estimates across all cases.

1.2.5 Conclusion and Future Directions

In this paper, we described *Little Ball of Fur* – an open-source Python graph sampling framework built on the widely used scientific computing libraries NetworkX (Hagberg, Swart and S Chult, 2008), NetworKit (Staudt, Sazonovs and Meyerhenke, 2016), and NumPy (Walt, Colbert and Varoquaux, 2011). In detail, it provides techniques for node, edge, and exploration-based graph

DOI: 10.14754/CEU.2024.08

sampling.

We reviewed the general conventions that we used for implementing graph sampling algorithms in *Little Ball of Fur*. The framework offers a limited number of public methods, ingests and outputs data in widely used graph formats, and embodies preset default hyperparameters. We presented the practical implications of these design features with illustrative examples of Python code snippets. Using various social networks and web graphs we had shown that using sampled graphs extracted with *Little Ball of Fur* one can approximate ground truth graph level statistics such as transitivity and the degree correlation coefficient. We also found evidence that sampling subgraphs with our framework can accelerate node and graph classification without extremely reducing the predictive performance.

As we have emphasized *Little Ball of Fur* assumes that the inputted graph is undirected and unweighted. In the future, we envision to relax these assumptions about the input. We plan to include additional high-performance backend libraries such as SNAP (Leskovec and Krevl, 2014) and GraphTool (Peixoto, 2014). Furthermore, we aim to extend our framework by including multiplex (Gjoka et al., 2011), attributed, and heterogeneous (Li and Yeh, 2011; Yang et al., 2013) graph sampling algorithms with new releases of the framework. Our library can be used to carry out comprehensive evaluation of different techniques to find the best method for different ground truth statistics, as showcased by Table 1.14. Future research can extend the set of statistics and networks considered in this analysis.

DOI: 10.14754/CEU.2024.08

1.3 Pytorch Geometric Temporal: Spatiotemporal signal processing with neural machine learning models

1.3.1 Introduction

Deep learning on static graph-structured data has seen unprecedented success in various business and scientific application domains. Neural network layers which operate on graph data can serve as building blocks of document labeling, fraud detection, traffic forecasting, and cheminformatics systems (Rozemberczki et al., 2021*a*; Rozemberczki, Kiss and Sarkar, 2020*a*; Yu, Yin and Zhu, 2018; Bojchevski et al., 2020; Rozemberczki et al., 2020). This emergence and the widespread adaptation of geometric deep learning was made possible by open-source machine learning libraries. The high quality, breadth, user-oriented nature, and availability of specialized deep learning libraries (Fey and Lenssen, 2019; Zheng et al., 2020*b*; Data61, 2018; Rozemberczki, Kiss and Sarkar, 2020*a*) were all contributing factors to the practical success and large-scale deployment of graph machine learning systems. At the same time, the existing geometric deep learning frameworks operate on graphs that have a fixed topology and it is also assumed that the node features and labels are static. Besides limiting assumptions about the input data, these off-the-shelf libraries are not designed to operate on spatiotemporal data.

We propose PyTorch Geometric Temporal, an open-source Python library for spatiotemporal machine learning. We designed PyTorch Geometric Temporal with a simple and consistent API inspired by the software architecture of existing widely used geometric deep learning libraries from the PyTorch ecosystem (Paszke et al., 2019; Fey and Lenssen, 2019). Our framework was built by applying simple design principles consistently. The framework reuses existing neural network layers in a modular manner, in which models have a limited number of public methods and hyperparameters can be inspected. Spatiotemporal signal iterators ingest data memory efficiently in widely used scientific computing formats and return those in a PyTorch compatible format. The design principles in combination with the test coverage, documentation, practical

DOI: 10.14754/CEU.2024.08

tutorials, continuous integration, package indexing, and frequent releases make the framework an end-user-friendly spatiotemporal machine learning system.

The experimental evaluation of the framework entails node-level regression tasks on datasets released exclusively with the framework. Specifically, we compare the predictive performance of spatiotemporal graph neural networks on epidemiological forecasting, demand planning, web traffic management, and social media interaction prediction tasks. Synthetic experiments show that with the right batching strategy PyTorch Geometric Temporal is highly scalable and benefits from GPU accelerated computing.

The main contributions of our work can be summarized as:

- We publicly release *PyTorch Geometric Temporal*, the first deep learning library for parametric spatiotemporal machine learning models.
- We provide data loaders and iterators with *PyTorch Geometric Temporal* which can handle spatiotemporal datasets.
- We release new spatiotemporal benchmark datasets from the renewable energy production, epidemiological reporting, goods delivery, and web traffic forecasting domains.
- We evaluate the spatiotemporal forecasting capabilities of the neural and parametric machine learning models available in *PyTorch Geometric Temporal* on real-world datasets.

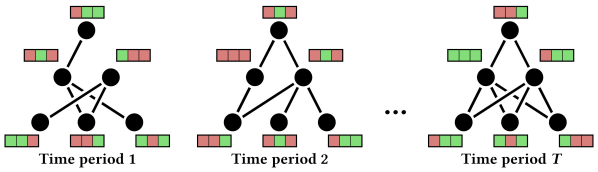
The remainder of the paper has the following structure. In Section 1.3.2 we overview important preliminaries and the related work about temporal and geometric deep learning and the characteristics of related open-source machine learning software. The main design principles of *PyTorch Geometric Temporal* are discussed in Section 1.3.3 with a practical example. We demonstrate the forecasting capabilities of the framework in Section 1.3.4 where we also evaluate the scalability of the library on various commodity hardware. We conclude in Section 1.3.5 where we summarize the results. The source code of *PyTorch Geometric Temporal* is publicly available at `https://github.com/benedekrozemberczki/pytorch_geometric_temporal`;

DOI: 10.14754/CEU.2024.08

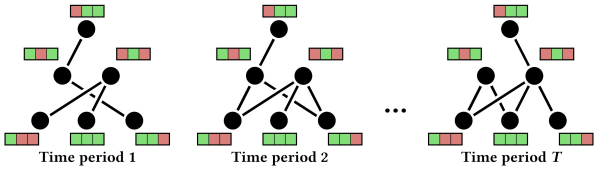
the Python package can be installed via the *Python Package Index*. Detailed documentation is accessible at `https://pytorch-geometric-temporal.readthedocs.io/`.

1.3.2 Preliminaries and related work

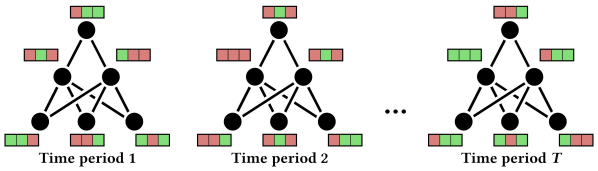
In order to position our contribution and highlight its significance, we introduce some important concepts about spatiotemporal data and discuss related literature about geometric deep learning and machine learning software.



(a) Dynamic graph with temporal signal.



(b) Dynamic graph with static signal.



(c) Static graph with temporal signal.

Figure 1.15: The data iterators in PyTorch Geometric Temporal can provide temporal snapshots for all of the non static geometric deep learning scenarios.

Temporal Graph Sequences

Our framework considers specific input data types on which the spatiotemporal machine learning models operate. Input data types can differ in terms of the dynamics of the graph and that of the modelled vertex attributes. We take a discrete temporal snapshot view of this data

DOI: 10.14754/CEU.2024.08

representation problem (Holme and Saramäki, 2012; Holme, 2015) and our work considers three spatiotemporal data types which can be described by the subplots of Figure 1.15 and the following formal definitions:

Definition 1 ***Dynamic graph with temporal signal*** *A dynamic graph with a temporal signal is the ordered set of graph and node feature matrix tuples* $\mathcal{D} = \{(\mathcal{G}_1, \mathbf{X}_1), \ldots, (\mathcal{G}_T, \mathbf{X}_T)\}$ *where the vertex sets satisfy that* $V_t = V,\ \forall t \in \{1, \ldots, T\}$ *and the node feature matrices that* $\mathbf{X}_t \in \mathbb{R}^{|V| \times d},\ \forall t \in \{1, \ldots, T\}$.

Definition 2 ***Dynamic graph with static signal.*** *A dynamic graph with a static signal is the ordered set of graph and node feature matrix tuples* $\mathcal{D} = \{(\mathcal{G}_1, \mathbf{X}), \ldots, (\mathcal{G}_T, \mathbf{X})\}$ *where vertex sets satisfy* $V_t = V, \forall t \in \{1, \ldots, T\}$ *and the node feature matrix that* $\mathbf{X} \in \mathbb{R}^{|V| \times d}$.

Definition 3 ***Static graph with temporal signal.*** *A static graph with a temporal signal is the ordered set of graph and node feature matrix tuples* $\mathcal{D} = \{(\mathcal{G}, \mathbf{X}_1), \ldots, (\mathcal{G}, \mathbf{X}_T)\}$ *where the node feature matrix satisfies that* $\mathbf{X}_t \in \mathbb{R}^{|V| \times d},\ \forall t \in \{1, \ldots, T\}$.

Representing spatiotemporal data based on these theoretical concepts allows us the creation of memory-efficient data structures which conceptualize these definitions in practice well.

Deep Learning with Time and Geometry

Our work provides deep learning models that operate on data that has both temporal and spatial aspects. These techniques are natural recombinations of existing neural network layers that operate on sequences and static graph-structured data.

Temporal Deep Learning A large family of temporal deep learning models such as the *LSTM* (Hochreiter and Schmidhuber, 1997) and *GRU* (Cho et al., 2014) generates in-memory representations of data points which are iteratively updated as it learns by new snapshots. Another family of deep learning models uses the *attention mechanism* (Luong, Pham and Manning, 2015;

DOI: 10.14754/CEU.2024.08

Bahdanau, Cho and Bengio, 2015; Vaswani et al., 2017) to learn representations of the data points which are adaptively recontextualized based on the temporal history. These types of models serve as templates for the temporal block of spatiotemporal deep learning models.

Static Graph Representation Learning Learning representations of vertices, edges, and whole graphs with graph neural networks in a supervised or unsupervised way can be described by the *message passing* formalism (Gilmer et al., 2017). In this conceptual framework using the node and edge attributes in a graph as parametric function generates compressed representations (messages) which are propagated between the nodes based on a message-passing rule and aggregated to form new representations. Most of the existing graph neural network architectures such as GCN (Kipf and Welling, 2017), GGCN (Li et al., 2016), ChebyConv (Defferrard, Bresson and Vandergheynst, 2016), and RGCN (Schlichtkrull et al., 2018) fit perfectly into this general description of graph neural networks. Models are differentiated by assumptions about the input graph (e.g. node heterogeneity, multiplexity, presence of edge attributes), the message compression function used, the propagation scheme, and the message aggregation function applied to the received messages.

Spatiotemporal Deep Learning A spatiotemporal deep learning model fuses the basic conceptual ideas of temporal deep learning techniques and graph representation learning. Operating on a temporal graph sequence, these models perform message-passing at each time point with a graph neural network block, and the new temporal information is incorporated by a temporal deep learning block. This design allows for sharing salient temporal and spatial autocorrelation information across the spatial units. The temporal and spatial layers which are fused together in a single parametric machine learning model are trained together jointly by exploiting the fact that the fused models are end-to-end differentiable. In Table 1.15 we summarized the spatiotemporal deep learning models implemented in the framework which we categorized based on the temporal and graph neural network layer blocks, the order of spatial proximity, and

DOI: 10.14754/CEU.2024.08

heterogeneity of the edge set.

Graph Representation Learning Software

The current graph representation learning software ecosystem which allows academic research and industrial deployment extends open-source auto-differentiation libraries such as TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2019), MxNet (Chen et al., 2015) and JAX (Bradbury et al., 2018; Frostig, Johnson and Leary, 2018). Our work does the same as we build on the PyTorch Geometric ecosystem. We summarize the characteristics of these libraries in Table 1.16 which enables for comparing frameworks based on the backend, presence of supervised training functionalities, presence of temporal models, and GPU support. Our proposed framework is the only one to date which allows the supervised training of temporal graph representation learning models with graphics card-based acceleration.

Spatiotemporal Data Analytics Software

The open-source ecosystem for spatiotemporal data processing consists of specialized database systems, basic analytical tools, and advanced machine learning libraries. We summarized the characteristics of the most popular libraries in Table 1.17 with respect to the year of release, the purpose of the framework, source code language, and GPU support.

First, it is evident that most spatiotemporal data processing tools are fairly new and there is much space for contributions in each subcategory. Second, the database systems are written in high-performance languages while the analytics and machine learning oriented tools have a pure Python/R design or a wrapper written in these languages. Finally, the use of GPU acceleration is not widespread which alludes to the fact that current spatiotemporal data processing tools might have a scalability issue. Our proposed framework PyTorch Geometric Temporal is the first fully open-source GPU accelerated spatiotemporal machine learning library for graph-structured data. Our experimental results showcase that GPU acceleration is greatly beneficial to the implemented algorithms. Therefore building our library in a way that it is compatible

DOI: 10.14754/CEU.2024.08

with pre-existing GPU-acceleration solutions implemented in Pytorch (Paszke et al., 2019) is a crucial feature of our work and a major improvement compared to other, state-of-the-art libraries providing similar functionalities.

1.3.3 The Framework design

Our primary goal is to give a general theoretical overview of the framework, discuss the framework design choices, give a detailed practical example and highlight our strategy for the long-term viability and maintenance of the project.

Neural Network Layer Design

The spatiotemporal neural network layers are implemented as classes in the framework. Each of the classes has a similar architecture driven by a few simple design principles.

Non-proliferation of classes The framework reuses the existing high-level neural network layer classes as building blocks from the PyTorch and PyTorch Geometric ecosystems. The goal of the library is not to replace the existing frameworks. This design strategy makes sure that the number of auxiliary classes in the framework is kept low and that the framework interfaces well with the rest of the ecosystem.

Hyperparameter inspection and type hinting The neural network layers do not have default hyperparameter settings as some of these have to be set in a dataset-dependent manner. In order to help with this, the layer hyperparameters are stored as public class attributes and are available for inspection. Moreover, the constructors of the neural network layers use type hinting which helps the end-users to set the hyperparameters.

Limited number of public methods The spatiotemporal neural network layers in our framework have a limited number of public methods for simplicity. For example, the auxiliary layer

DOI: 10.14754/CEU.2024.08

initialization methods and other internal model mechanics are implemented as private methods. All of the layers provide a forward method and those which explicitly use the *message-passing* scheme in PyTorch Geometric provide a public *message* method.

Auxiliary layers The auxiliary neural network layers which are not part of the *PyTorch Geometric* ecosystem such as diffusion convolutional graph neural networks (Li et al., 2018) are implemented as standalone neural network layers in the framework. These layers are available for the design of novel neural network architectures as individual components.

Data Structures

The design of *PyTorch Geometric Temporal* required the introduction of custom data structures that can efficiently store the datasets and provide temporally ordered snapshots for batching.

Spatiotemporal Signal Iterators Based on the categorization of spatiotemporal signals discussed in Section 1.3.2 we implemented three types of *Spatiotemporal Signal Iterators*. These iterators store spatiotemporal datasets in memory efficiently without redundancy. For example, a *Static Graph Temporal Signal* iterator will not store the edge indices and weights for each time period in order to save memory. By iterating over a *Spatiotemporal Signal Iterator* at each step a graph snapshot is returned which describes the graph of interest at a given point in time. Graph snapshots are returned in temporal order by the iterators. The *Spatiotemporal Signal Iterators* can be indexed directly to access a specific graph snapshot – a design choice that facilitates the use of advanced temporal batching.

Graph Snapshots The time period specific snapshots which consist of labels, features, edge indices and weights are stored as *NumPy* arrays (Van Der Walt, Colbert and Varoquaux, 2011) in memory, but returned as a *PyTorch Geometric Data* object instance (Fey and Lenssen, 2019) by the *Spatiotemporal Signal Iterators* when these are iterated on. This design choice hedges

DOI: 10.14754/CEU.2024.08

against the proliferation of classes and exploits the existing and widely used compact data structures from the *PyTorch* ecosystem (Paszke et al., 2019).

Train-Validation-Test Splitting As part of the library we provide a temporal train-test splitting function that creates train and test snapshot iterators from a Spatiotemporal Signal Iterator given a test dataset ratio. This parameter of the splitting function decides the fraction of data that is separated from the end of the spatiotemporal graph snapshot sequence for testing. The returned iterators have the same type as the input iterator. Importantly, this splitting does not influence the applicability of widely used semi-supervised model training strategies such as node masking.

Integrated Benchmark Dataset Loaders We provided easy-to-use practical data loader classes for widely used existing (Panagopoulos, Nikolentzos and Vazirgiannis, 2021) and the newly released benchmark datasets. These loaders return *Spatiotemporal Signal Iterators* which can be used for training existing and custom-designed spatiotemporal neural network architectures to solve supervised machine learning problems.

Design in Practice Case Study: Cumulative Model Training on CPU

In the following, we overview a simple end-to-end machine learning pipeline designed with PyTorch Geometric Temporal. These code snippets solve a practical epidemiological forecasting problem – predicting the weekly number of chickenpox cases in Hungary (Rozemberczki et al., 2021*a*). The pipeline consists of data preparation, model definition, training, and evaluation phases.

```
from torch_geometric_temporal import ChickenpoxDatasetLoader
from torch_geometric_temporal import temporal_signal_split

loader = ChickenpoxDatasetLoader()
```

DOI: 10.14754/CEU.2024.08

```
5
6 dataset = loader.get_dataset()
7
8 train, test = temporal_signal_split(dataset,
9                                      train_ratio=0.9)
```

Listings 1.1: Loading a default benchmark dataset and creating a temporal split with PyTorch Geometric Temporal.

Dataset Loading and Splitting In Listings 1.1 as a first step we import the Hungarian chickenpox cases benchmark dataset loader and the temporal train test splitter function (lines 1-2). We define the dataset loader (line 4) and use the *get_dataset()* class method to return a temporal signal iterator (line 5). Finally, we create a train-test split of the spatiotemporal dataset by using the splitting function and retain 10% of the temporal snapshots for model performance evaluation (lines 7-8).

```
1 import torch
2 import torch.nn.functional as F
3 from torch_geometric_temporal.nn.recurrent import DCRNN
4
5 class RecurrentGCN(torch.nn.Module):
6     def __init__(self, node_features, filters):
7         super(RecurrentGCN, self).__init__()
8         self.recurrent = DCRNN(node_features, filters, 1)
9         self.linear = torch.nn.Linear(filters, 1)
10
11    def forward(self, x, edge_index, edge_weight):
12        h = self.recurrent(x, edge_index, edge_weight)
13        h = F.relu(h)
```

DOI: 10.14754/CEU.2024.08

```
14         h = F.dropout(h, training=self.training)
15         h = self.linear(h)
16         return h
```

Listings 1.2: Defining a recurrent graph convolutonal neural network using PyTorch Geometric Temporal consisting of a diffusion convolutional spatiotemporal layer followed by rectified linear unit activation, dropout and a feedforward neural network layer.

Recurrent Graph Convolutional Model Definition We define a recurrent graph convolutional neural network model in Listings 1.2. We import the base and functional programming PyTorch libraries and one of the neural network layers from PyTorch Geometric Temporal (lines 1-3). The model requires a node feature count and convolutional filter parameter in the constructor (line 6). The model consists of a one-hop Diffusion Convolutional Recurrent Neural Network layer (Li et al., 2018) and a fully connected layer with a single neuron (lines 8-9).

In the forward pass method of the neural network, the model uses the vertex features, edges, and optional edge weights (line 11). The initial recurrent graph convolution-based aggregation (line 12) is followed by a rectified linear unit activation function (Nair and Hinton, 2010) and dropout (Srivastava et al., 2014) for regularization (lines 13-14). Using the fully connected layer the model outputs a single score for each spatial unit (lines 15-16).

```
1 model = RecurrentGCN(node_features=8, filters=32)
2
3 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
4
5 model.train()
6
7 for epoch in range(200):
8     cost = 0
9     for time, snapshot in enumerate(train):
```

```
10          y_hat = model(snapshot.x,
11                        snapshot.edge_index,
12                        snapshot.edge_attr)
13          cost = cost + torch.mean((y_hat-snapshot.y)**2)
14      cost = cost / (time+1)
15      cost.backward()
16      optimizer.step()
17      optimizer.zero_grad()
```

Listings 1.3: Creating a recurrent graph convolutional neural network and training it by cumulative weight updates.

Model Training Using the dataset split and the model definition we can turn our attention to training a regressor. In Listings 1.3 we create a model instance (line 1), transfer the model parameters (line 3) to the Adam optimizer (Kingma and Ba, 2015) which uses a learning rate of 0.01 and set the model to be trainable (line 5). In each epoch, we set the accumulated cost to be zero (line 8), iterate over the temporal snapshots in the training data (line 9), make forward passes with the model on each temporal snapshot, and accumulate the spatial unit-specific mean squared errors (lines 10-13). We normalize the cost, backpropagate and update the model parameters (lines 14-17).

Table 1.14: Ground truth and estimated descriptive statistics of the web graphs and social networks. We calculated average statistics from 10 seeded experimental runs and included the standard errors below the mean. We included the ground truth values based on the whole graph (first block) with estimates obtained with node (second block), edge (third block) and exploration (fourth and fifth blocks) sampling algorithms. Bold numbers denote for each category the best estimate for a given dataset.

	Facebook Page-Page			**Wikipedia Crocodiles**			**LastFM Asia**			**Deezer Hungary**		
	Clustering Coefficient	**Average Degree**	**Degree Correlation**	**Clustering Coefficient**	**Average Degree**	**Degree Correlation**	**Clustering Coefficient**	**Average Degree**	**Degree Correlation**	**Clustering Coefficient**	**Average Degree**	**Degree Correlation**
Truth	0.232	15.205	0.085	0.026	29.365	−0.277	0.179	7.294	0.017	0.093	9.377	0.207
RN Stumpf, Wiuf and May (2005)	**0.229** 0.002	7.531 0.060	**0.070** 0.003	**0.028** 0.001	14.293 0.388	**-0.284** 0.008	**0.177** 0.004	3.642 0.032	**0.020** 0.010	**0.092** 0.001	4.699 0.010	0.190 0.002
DRN Adamic et al. (2001)	0.261 0.001	21.514 0.021	0.080 0.001	0.038 0.001	40.750 0.054	−0.324 0.001	0.231 0.001	9.531 0.020	0.045 0.001	0.102 0.001	**8.551** 0.007	**0.211** 0.001
PRN Leskovec and Faloutsos (2006)	0.270 0.001	**16.228** 0.032	0.136 0.001	0.049 0.001	**32.370** 0.074	−0.290 0.001	0.236 0.001	**8.209** 0.022	0.064 0.002	0.098 0.001	7.251 0.007	0.231 0.001
RE Krishnamurthy et al. (2005)	0.116 0.001	8.470 0.004	**0.084** 0.001	0.013 0.001	15.462 0.009	**-0.277** 0.001	0.090 0.001	4.422 0.009	**0.019** 0.002	0.046 0.001	5.018 0.001	0.183 0.001
RNE Krishnamurthy et al. (2005)	0.092 0.001	7.602 0.001	−0.075 0.001	0.007 0.001	14.682 0.001	−0.231 0.001	0.046 0.001	3.674 0.001	−0.108 0.001	0.042 0.001	4.701 0.001	0.056 0.001
HRNE Krishnamurthy et al. (2005)	0.081 0.001	7.194 0.002	−0.005 0.001	0.007 0.001	13.550 0.005	−0.234 0.001	0.046 0.001	3.562 0.003	−0.070 0.002	0.039 0.001	4.501 0.001	0.095 0.001
TIES Ahmed, Neville and Kompella (2013)	0.235 0.001	16.564 0.007	0.083 0.001	**0.026** 0.001	30.720 0.016	−0.278 0.001	0.190 0.001	8.218 0.010	0.027 0.001	**0.094** 0.001	**9.748** 0.001	0.204 0.001
PIES Ahmed, Neville and Kompella (2013)	**0.231** 0.001	**15.357** 0.008	0.087 0.001	**0.026** 0.001	**29.142** 0.015	−0.283 0.001	**0.186** 0.001	**7.247** 0.010	0.037 0.001	0.086 0.001	8.501 0.003	**0.209** 0.001
RW Gjoka et al. (2010)	0.255 0.001	22.535 0.022	0.073 0.001	0.036 0.001	41.648 0.196	−0.325 0.001	0.224 0.001	9.878 0.021	0.039 0.003	0.104 0.001	9.221 0.008	0.218 0.001
RWR Leskovec and Faloutsos (2006)	0.253 0.003	20.282 0.293	0.092 0.007	0.043 0.003	38.967 0.549	−0.313 0.006	0.222 0.003	9.078 0.087	0.036 0.004	0.098 0.001	9.122 0.082	**0.213** 0.003
RWJ Ribeiro and Towsley (2010)	0.271 0.001	18.615 0.036	0.123 0.001	0.047 0.001	34.475 0.074	−0.297 0.001	0.233 0.001	9.012 0.032	0.067 0.003	0.102 0.001	8.351 0.016	0.233 0.002
MHRW Hübler et al. (2008); Stutzbach et al. (2008)	0.280 0.002	**17.903** 0.113	0.134 0.003	0.119 0.002	**29.914** 0.223	−0.146 0.004	0.232 0.001	**8.854** 0.041	0.102 0.007	0.106 0.001	7.761 0.023	0.242 0.003
RC-MHRW Li et al. (2015)	0.266 0.001	21.070 0.064	0.106 0.002	0.072 0.001	35.758 0.159	−0.254 0.002	0.232 0.001	9.594 0.031	0.078 0.003	0.103 0.001	**8.553** 0.021	0.237 0.002
FRW Ribeiro and Towsley (2010)	0.063 0.001	5.745 0.059	0.069 0.004	0.004 0.001	5.813 0.058	**-0.280** 0.003	0.084 0.001	4.485 0.032	**0.018** 0.008	0.033 0.001	3.243 0.005	0.091 0.002
CNRW Zhou, Zhang and Das (2015)	0.255 0.001	22.590 0.038	0.072 0.001	0.037 0.001	41.645 0.104	−0.324 0.001	0.223 0.001	9.924 0.018	0.036 0.002	0.104 0.001	9.254 0.017	0.218 0.001
CNARW Li et al. (2019*c*)	**0.239** 0.001	21.117 0.033	**0.082** 0.001	**0.026** 0.001	41.064 0.101	−0.348 0.001	**0.220** 0.001	9.508 0.032	0.052 0.002	**0.094** 0.001	9.140 0.013	0.218 0.001
NBT-RW Lee, Xu and Eun (2012)	0.257 0.001	22.353 0.048	0.076 0.001	0.038 0.001	41.264 0.144	−0.322 0.001	0.226 0.001	9.818 0.027	0.049 0.002	0.104 0.001	9.106 0.010	0.230 0.001
SB Goodman (1961)	**0.238** 0.002	20.671 0.223	0.069 0.004	0.057 0.004	37.278 0.576	−0.292 0.009	0.207 0.002	9.131 0.121	−0.008 0.005	**0.093** 0.001	**9.913** 0.103	0.122 0.003
FF Leskovec, Kleinberg and Faloutsos (2005)	**0.238** 0.001	19.219 0.089	**0.079** 0.002	0.074 0.002	**33.190** 0.262	−0.227 0.006	0.204 0.001	9.034 0.025	0.051 0.001	0.096 0.001	10.120 0.013	0.197 0.001
CSE Maiya and Berger-Wolf (2010)	0.229 0.002	**13.116** 0.046	0.070 0.003	**0.026** 0.001	20.314 0.345	**-0.290** 0.006	**0.191** 0.003	6.544 0.055	**0.006** 0.006	0.089 0.002	6.554 0.001	0.165 0.001
SP Rezvanian and Meybodi (2015)	0.221 0.001	12.842 0.062	0.106 0.002	0.037 0.001	23.451 0.125	−0.292 0.001	0.203 0.001	**7.258** 0.032	0.043 0.002	0.079 0.001	8.176 0.007	**0.209** 0.001

DOI: 10.14754/CEU.2024.08

Table 1.15: A comparison of spatiotemporal deep learning models in PyTorch Geometric Temporal based on the temporal and spatial block, proximity order and edge heterogeneity.

Model	**Temporal Layer**	**GNN Layer**	**Proximity Order**	**Multi Type**
DCRNN (Li et al., 2018)	GRU	DiffConv	Higher	False
GConvGRU (Seo et al., 2018)	GRU	Chebyshev	Lower	False
GConvLSTM (Seo et al., 2018)	LSTM	Chebyshev	Lower	False
GC-LSTM (Chen et al., 2018*b*)	LSTM	Chebyshev	Lower	True
DyGrAE (Taheri and Berger-Wolf, 2019; Taheri, Gimpel and Berger-Wolf, 2019)	LSTM	GGCN	Higher	False
LRGCN (Li et al., 2019*a*)	LSTM	RGCN	Lower	False
EGCN-H (Pareja et al., 2020)	GRU	GCN	Lower	False
EGCN-O (Pareja et al., 2020)	LSTM	GCN	Lower	False
T-GCN (Zhao et al., 2019)	GRU	GCN	Lower	False
A3T-GCN (Zhu et al., 2020)	GRU	GCN	Lower	False
AGCRN (Bai et al., 2020)	GRU	Chebyshev	Higher	False
MPNN LSTM (Panagopoulos, Nikolentzos and Vazirgiannis, 2021)	LSTM	GCN	Lower	False
STGCN (Yu, Yin and Zhu, 2018)	Attention	Chebyshev	Higher	False
ASTGCN (Guo et al., 2019)	Attention	Chebyshev	Higher	False
MSTGCN (Guo et al., 2019)	Attention	Chebyshev	Higher	False
GMAN (Zheng et al., 2020*a*)	Attention	Custom	Lower	False
MTGNN (Wu et al., 2020)	Attention	Custom	Higher	False
AAGCN (Shi et al., 2019)	Attention	Custom	Higher	False
DNNTSP (Yu et al., 2020)	Attention	GCN	Lower	False

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

Table 1.16: A desiderata and backend based comparison of open-source geometric deep learning libraries.

Library	Backend	Supervised	Temporal	GPU
PT Geometric (Fey and Lenssen, 2019)	PT	✔	✘	✔
Geometric2DR (Scherer and Lio, 2020)	PT	✘	✘	✔
CogDL (Cen et al., 2021)	PT	✔	✘	✔
Spektral (Grattarola and Alippi, 2020)	TF	✔	✘	✔
TF Geometric (Hu et al., 2021)	TF	✔	✘	✔
StellarGraph (Data61, 2018)	TF	✔	✘	✔
DGL (Zheng et al., 2020*b*)	TF/PT/MX	✔	✘	✔
DIG (Liu et al., 2021*a*)	PT	✔	✘	✔
Jraph (Godwin* et al., 2020)	JAX	✔	✘	✔
Graph-Learn (Yang, 2019)	Custom	✔	✘	✔
GEM (Goyal et al., 2018)	TF	✘	✘	✔
DynamicGEM (Goyal and Ferrara, 2018)	TF	✘	✔	✔
OpenNE (Tu et al., 2018)	Custom	✘	✘	✘
Karate Club (Rozemberczki, Kiss and Sarkar, 2020*a*)	Custom	✘	✘	✘
Our Work	PT	✔	✔	✔

Table 1.17: A multi-aspect comparison of open-source spatiotemporal database systems, data analytics libraries and machine learning frameworks.

Library	Year	Purpose	Language	GPU
GeoWave (Whitby, Fecher and Bennight, 2017)	2016	Database	Java	✘
StacSpec (Hanson, 2019)	2017	Database	Javascript	✘
MobilityDB (Zimányi, Sakr and Lesuisse, 2020)	2019	Database	C	✘
PyStac (Rob, 2020)	2020	Database	Python	✘
StaRs (Pebesma, 2017)	2017	Analytics	R	✘
CuSpatial (Taylor et al., 2019)	2019	Analytics	Python	✔
PySAL (Rey and Anselin, 2010)	2017	Machine Learning	Python	✘
STDMTMB (Anderson et al., 2018)	2018	Machine Learning	R	✘
Our work	2021	Machine Learning	Python	✔

DOI: 10.14754/CEU.2024.08

```
model.eval()
cost = 0
for time, snapshot in enumerate(test):
    y_hat = model(snapshot.x,
                  snapshot.edge_index,
                  snapshot.edge_attr)
    cost = cost + torch.mean((y_hat-snapshot.y)**2)
cost = cost / (time+1)
cost = cost.item()
print("MSE: {:.4f}".format(cost))
```

Listings 1.4: Evaluating the recurrent graph convolutional neural network on the test portion of the spatiotemporal dataset using the time unit averaged mean squared error.

Model Evaluation The scoring of the trained recurrent graph neural network in Listings 1.4 uses the snapshots in the test dataset. We set the model to be non-trainable and the accumulated squared errors as zero (lines 1-2). We iterate over the test spatiotemporal snapshots, make forward passes to predict the number of chickenpox cases, and accumulate the squared error (lines 3-7). The accumulated errors are normalized and we can print the mean squared error calculated on the whole test horizon (lines 8-10).

Design in Practice Case Study: Incremental Model Training with GPU Acceleration

Exploiting the power of GPU-based acceleration of computations happens at the training and evaluation steps of the PyTorch Geometric Temporal pipelines. In this case study, we assume that the Hungarian Chickenpox cases dataset is already loaded in memory, the temporal split happened and a model class was defined by the code snippets in Listings 1.1 and 1.2. Moreover,

we assume that the machine used for training the neural network can access a single CUDA compatible GPU device Sanders and Kandrot (2010).

```
1 model = RecurrentGCN(node_features=8, filters=32)
2 device = torch.device('cuda')
3 model = model.to(device)
4
5 optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
6 model.train()
7
8 for epoch in range(200):
9     for snapshot in train:
10         snapshot = snapshot.to(device)
11         y_hat = model(snapshot.x,
12                       snapshot.edge_index,
13                       snapshot.edge_attr)
14         cost = torch.mean((y_hat-snapshot.y)**2)
15         cost.backward()
16         optimizer.step()
17         optimizer.zero_grad()
```

Listings 1.5: Creating a recurrent graph convolutional neural network instance and training it by incremental weight updates on a GPU.

Model Training In Listings 1.5 we demonstrate accelerated training with incremental weight updates. The model of interest and the device used for training are defined while the model is transferred to the GPU (lines 1-3). The optimizer registers the model parameters and the model parameters are set to be trainable (lines 5-6). We iterate over the temporal snapshot iterator 200 times and the iterator returns a temporal snapshot in each step. Importantly the snapshots

DOI: 10.14754/CEU.2024.08

which are PyTorch Geometric Data objects are transferred to the GPU (lines 8-10). The use of PyTorch Geometric Data objects as temporal snapshots enables the transfer of the time period specific edges, node features, and target vector with a single command. Using the input data a forward pass is made, the loss is accumulated and weight updates happen using the optimizer in each time period (lines 11-17). Compared to the cumulative backpropagation-based training approach discussed in subsubsection 1.3.3 this backpropagation strategy is slower as weight updates happen at each time step, not just at the end of training epochs.

Model Evaluation During model scoring the GPU can be utilized again. The snippet in Listings 1.6 demonstrates that the only modification needed for accelerated evaluation is the transfer of snapshots to the GPU. In each time period, we move the temporal snapshot to the device to do the forward pass (line 4). We do the forward pass with the model and the snapshot on the GPU and accumulate the loss (lines 5-8). The loss value is averaged out and detached from the GPU for printing (lines 9-11).

```
1 model.eval()
2 cost = 0
3 for time, snapshot in enumerate(test):
4     snapshot = snapshot.to(device)
5     y_hat = model(snapshot.x,
6                   snapshot.edge_index,
7                   snapshot.edge_attr)
8     cost = cost + torch.mean((y_hat-snapshot.y)**2)
9 cost = cost / (time+1)
10 cost = cost.item()
11 print("MSE: {:.4f}".format(cost))
```

Listings 1.6: Evaluating the recurrent graph convolutional neural network with GPU based acceleration.

DOI: 10.14754/CEU.2024.08

Maintaining PyTorch Geometric Temporal

The viability of the project is made possible by the open-source code, version control, public releases, automatically generated documentation, continuous integration, and nearly 100% test coverage.

Open-Source Code-Base and Public Releases The source code of *PyTorch Geometric Temporal* is publicly available on *GitHub* under the MIT license. Using an open version control system allowed us to have a large group collaborate on the project and have external contributors who also submitted feature requests. The public releases of the library are also made available on the *Python Package Index*, which means that the framework can be installed via the *pip* command using the terminal.

Documentation The source-code of *PyTorch Geometric Temporal* and *Sphinx* (Brandl, 2010) are used to generate a publicly available documentation of the library[2]. This documentation is automatically created every time when the code-base changes in the public repository. The documentation covers the constructors and public methods of neural network layers, temporal signal iterators, public dataset loaders, and splitters. It also includes a list of relevant research papers, an in-depth installation guide, a detailed getting-started tutorial, and a list of integrated benchmark datasets.

Continuous Integration We provide continuous integration for *PyTorch Geometric Temporal* with *GitHub Actions* which are available for free on *GitHub* without limitations on the number of builds. When the code is updated on any branch of the *GitHub* repository the build process is triggered and the library is deployed on *Linux*, *Windows* and *macOS* virtual machines.

Unit Tests and Code Coverage The temporal graph neural network layers, custom data structures, and benchmark dataset loaders are all covered by unit tests. These unit tests can be

[2]Available at `https://pytorch-geometric-temporal.readthedocs.io/`

DOI: 10.14754/CEU.2024.08

executed locally using the source code. Unit tests are also triggered by the continuous integration provided by *GitHub Actions*. When the master branch of the open-source *GitHub* repository is updated, the build is successful, and all of the unit tests pass a coverage report is generated by *CodeCov*.

1.3.4 Experimental evaluation

The proposed framework is evaluated on node-level regression tasks using novel datasets which we release with the paper. We also evaluate the effect of various batching techniques on predictive performance and runtime.

New Datasets

We release new spatiotemporal benchmark datasets with *PyTorch Geometric Temporal* which can be used to test models on node-level regression tasks. The descriptive statistics and properties of these newly introduced benchmark datasets are summarized in Table 1.18.

These newly released datasets are the following:

- **Chickenpox Hungary.** A spatiotemporal dataset about the officially reported cases of chickenpox in Hungary. The nodes are counties and edges describe direct neighborhood relationships. The dataset covers the weeks between 2005 and 2015 without missingness.

- **Windmill Output Datasets.** An hourly windfarm energy output dataset covering 2 years from a European country. Edge weights are calculated from the proximity of the windmills – high weights imply that two windmill stations are in close vicinity. The size of the dataset relates to the grouping of wind farms considered; the smaller datasets are more localized to a single region.

- **Pedal Me Deliveries.** A dataset about the number of weekly bicycle package deliveries by Pedal Me in London during 2020 and 2021. Nodes in the graph represent geographical

DOI: 10.14754/CEU.2024.08

units and edges are proximity-based mutual adjacency relationships.

- **Wikipedia Math.** Contains Wikipedia pages about popular mathematics topics and edges describe the links from one page to another. Features describe the number of daily visits between 2019 March and 2021 March.

- **Twitter Tennis RG and UO.** Twitter mention graphs of major tennis tournaments from 2017. Each snapshot contains the graph of popular player or sport news accounts and mentions between them (Béres et al., 2018, 2019). Node labels encode the number of mentions received and vertex features are structural properties.

- **Covid19 England.** A dataset about mass mobility between regions in England and the number of confirmed COVID-19 cases from March to May 2020 (Panagopoulos, Nikolentzos and Vazirgiannis, 2021). Each day contains a different mobility graph and node features corresponding to the number of cases in the previous days. Mobility stems from Facebook Data For Good [3] and cases from gov.uk [4].

- **Montevideo Buses.** A dataset about the hourly passenger inflow at bus stop level for eleven bus lines from the city of Montevideo. Nodes are bus stops and edges represent connections between the stops; the dataset covers a whole month of traffic patterns.

- **MTM-1 Hand Motions.** A temporal dataset of Methods-Time Measurement-1 (Maynard, Stegemerten and Schwab, 1948) motions, signaled as consecutive graph frames of 21 3D hand key points that were acquired via MediaPipe Hands (Zhang et al., 2020) from original RGB-Video material. Node features encode the normalized 3D-coordinates of each finger joint and the vertices are connected according to the human hand structure.

[3] https://dataforgood.fb.com/
[4] https://coronavirus.data.gov.uk/

DOI: 10.14754/CEU.2024.08

Table 1.18: Properties and granularity of the spatiotemporal datasets introduced in the paper with information about the number of time periods (T) and spatial units ($|V|$).

Dataset	Signal	Graph	Frequency	T	$\|V\|$
Chickenpox Hungary	Temporal	Static	Weekly	522	20
Windmill Large	Temporal	Static	Hourly	17,472	319
Windmill Medium	Temporal	Static	Hourly	17,472	26
Windmill Small	Temporal	Static	Hourly	17,472	11
Pedal Me Deliveries	Temporal	Static	Weekly	36	15
Wikipedia Math	Temporal	Static	Daily	731	1,068
Twitter Tennis RG	Static	Dynamic	Hourly	120	1000
Twitter Tennis UO	Static	Dynamic	Hourly	112	1000
Covid19 England	Temporal	Dynamic	Daily	61	129
Montevideo Buses	Temporal	Static	Hourly	744	675
MTM-1 Hand Motions	Temporal	Static	1/24 Seconds	14,469	21

Predictive Performance

The forecasting experiments focus on the evaluation of the recurrent graph neural networks implemented in our framework. We compare the predictive performance under two specific backpropagation regimes which can be used to train these recurrent models:

- **Incremental:** After each temporal snapshot the loss is backpropagated and model weights are updated. This would need as many weight updates as the number of temporal snapshots.
- **Cumulative:** When the loss from every temporal snapshot is aggregated, it is backpropagated, and weights are updated with the optimizer. This requires only one weight update step per epoch.

Experimental settings Using 90% of the temporal snapshots for training, we evaluated the forecasting performance on the last 10% by calculating the average mean squared error from 10 experimental runs. We used models with a recurrent graph convolutional layer which had 32 convolutional filters. The spatiotemporal layer was followed by the rectified linear unit (Nair and Hinton, 2010) activation function and during training time we used a dropout of 0.5 for regularization (Srivastava et al., 2014) after the spatiotemporal layer. The hidden representations were fed to a fully connected feedforward layer which outputted the predicted scores for each

DOI: 10.14754/CEU.2024.08

spatial unit. The recurrent models were trained for 100 epochs with the Adam optimizer (Kingma and Ba, 2015) which used a learning rate of 10^{-2} to minimize the mean squared error. Our target variables are normalized.

Experimental findings Results are presented in Table 1.19 where we also report standard deviations around the test set mean squared error and bold numbers denote the best performing model under each training regime on a dataset. Since our outcome variables are normalized, the error rates can be interpreted in units of variance. Our experimental findings demonstrate multiple important empirical regularities which have important practical implications, namely:

1. Most recurrent graph neural networks have a similar predictive performance on these regression tasks. In simple terms, there is not a single model which acts as a *silver bullet*. This also postulates that the model with the lowest training time is likely to be as good as the slowest one.

2. Results on the Wikipedia Math dataset imply that a cumulative backpropagation strategy can have a detrimental effect on the predictive performance of a recurrent graph neural network. When computation resources are not a bottleneck, an incremental strategy can be significantly better.

Runtime Performance

The evaluation of the PyTorch Geometric Temporal runtime performance focuses on manipulating the input size and measuring the time needed to complete a training epoch. We investigate the runtime under the incremental and cumulative backpropagation strategies.

Experimental settings The runtime evaluation used the GConvGRU model (Seo et al., 2018) with the hyperparameter settings described in subsubsection 1.3.4. We measured the time needed for doing a single epoch over a sequence of 100 synthetic graphs. Reference Watts-Strogatz graphs in the snapshots of the dynamic graph with temporal signal iterator have binary

DOI: 10.14754/CEU.2024.08

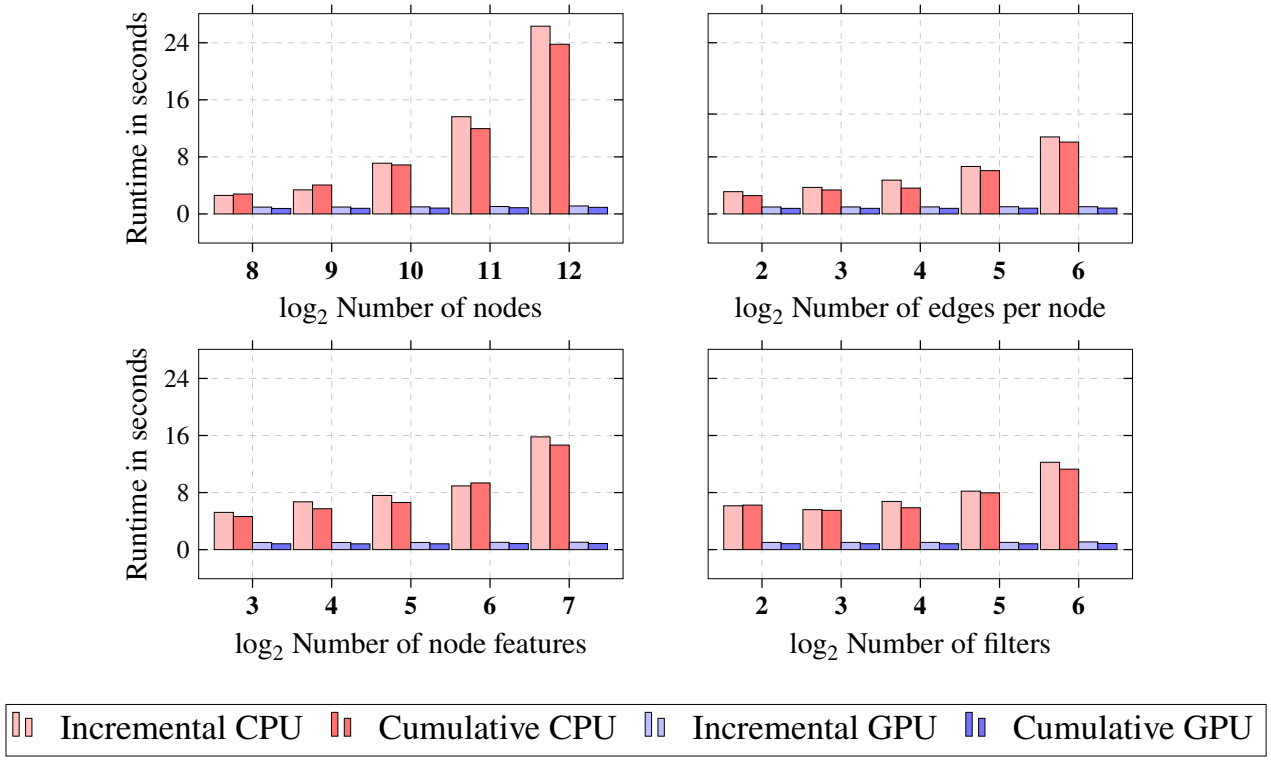


Figure 1.16: The average time needed for doing an epoch on a dynamic graph – temporal signal iterator of Watts Strogatz graphs with a recurrent graph convolutional model.

labels, 2^{10} nodes, 2^5 edges per node, and 2^5 node features. Runtimes were measured on both of the following hardware specifications:

- **CPU:** The machine used for benchmarking had 8 *Intel 1.00 GHz i5-1035G1* processors.
- **GPU:** We utilized a machine with a single *Tesla V-100* graphics card for the experiments.

Experimental findings We plot the average runtime calculated from 10 experimental runs on Figure 1.16 for each input size. We also include a log-transformed version of the graph depicting GPU results only on Figure 1.17 to better visualize the patterns in GPU runtimes. Our results about runtime have two important implications about the practical application of our framework:

1. The use of a cumulative backpropagation strategy only results in marginal computation gains compared to the incremental one.

DOI: 10.14754/CEU.2024.08

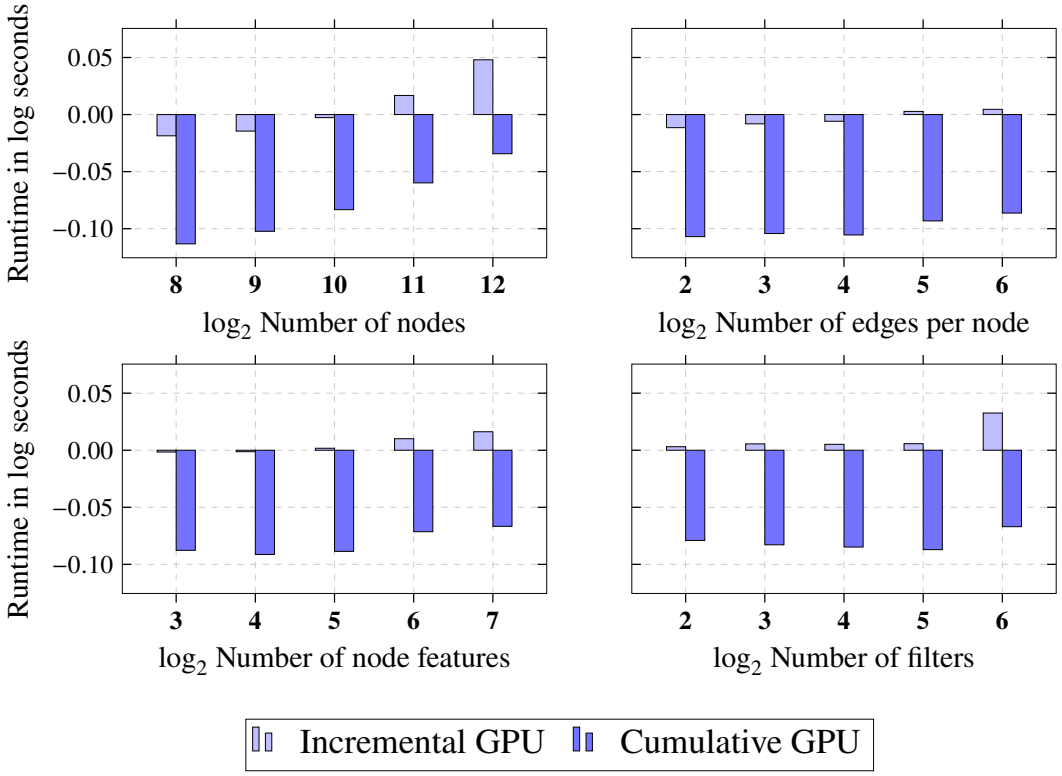


Figure 1.17: The average time needed for doing an epoch on a dynamic graph – temporal signal iterator of Watts Strogatz graphs with a recurrent graph convolutional model - GPU results only

2. On temporal sequences of large dynamically changing graphs the GPU-aided training can reduce the time needed to do an epoch by a whole magnitude. This result further emphasizes the importance of the GPU acceleration that is available in our library.

3. CPU and GPU training times present the same patterns across covariates. Increasing the network size clearly increases the runtime. This positive association holds for the number of node features, the network density and the number of filters as well.

1.3.5 Conclusions and Future Directions

In this paper, we discussed *PyTorch Geometric Temporal*, the first deep learning library designed for neural spatiotemporal signal processing. We reviewed the existing geometric deep learning and machine learning techniques implemented in the framework. We gave an overview of the

DOI: 10.14754/CEU.2024.08

general machine learning framework design principles, the newly introduced input, and output data structures, long-term project viability and discussed a case study with source code that utilized the library. Our empirical evaluation focused on (a) the predictive performance of the models available in the library on real-world datasets which we released with the framework; (b) the scalability of the methods under various input sizes and structures.

Our work could be extended and it also opens up opportunities for novel geometric deep learning and applied machine learning research. A possible direction to extend our work would be the consideration of continuous-time or time differences between temporal snapshots which are not constant. Another opportunity is the inclusion of temporal models which operate on curved spaces such as hyperbolic and spherical spaces. We are particularly interested in how the spatiotemporal deep learning techniques in the framework can be deployed and used for solving high-impact practical machine learning tasks.

Table 1.19: The predictive performance of spatiotemporal neural networks evaluated by average mean squared error. We report average performances calculated from 10 experimental repetitions with standard deviations around the average mean squared error calculated on 10% forecasting horizons. We use the incremental and cumulative backpropagation strategies. Bold numbers denote the best performance on each dataset given a training approach.

	Chickenpox Hungary		**Twitter Tennis RG**		**PedalMe London**		**Wikipedia Math**	
	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative
DCRNN (Li et al., 2018)	1.124 ± 0.015	1.123 ± 0.014	2.049 ± 0.023	2.043 ± 0.016	1.463 ± 0.019	1.450 ± 0.024	0.679 ± 0.020	**0.803 ± 0.018**
GConvGRU (Seo et al., 2018)	1.128 ± 0.011	1.132 ± 0.023	2.051 ± 0.020	2.007 ± 0.022	1.622 ± 0.032	1.944 ± 0.013	**0.657 ± 0.015**	0.837 ± 0.021
GConvLSTM (Seo et al., 2018)	1.121 ± 0.014	1.119 ± 0.022	2.049 ± 0.024	2.007 ± 0.012	1.442 ± 0.028	1.433 ± 0.020	0.777 ± 0.021	0.868 ± 0.018
GC-LSTM (Chen et al., 2018*b*)	1.115 ± 0.014	1.116 ± 0.023	2.053 ± 0.024	2.032 ± 0.015	**1.455 ± 0.023**	1.468 ± 0.025	0.779 ± 0.023	0.852 ± 0.016
DyGrAE (Taheri and Berger-Wolf, 2019; Taheri, Gimpel and Berger-Wolf, 2019)	1.120 ± 0.021	1.118 ± 0.015	**2.031 ± 0.006**	2.007 ± 0.004	**1.455 ± 0.031**	1.456 ± 0.019	0.773 ± 0.009	0.816 ± 0.016
EGCN-H (Pareja et al., 2020)	**1.113 ± 0.016**	**1.104 ± 0.024**	2.040 ± 0.018	**2.006 ± 0.008**	1.467 ± 0.026	1.436 ± 0.017	0.775 ± 0.022	0.857 ± 0.022
EGCN-O (Pareja et al., 2020)	1.124 ± 0.009	1.119 ± 0.020	2.055 ± 0.020	2.010 ± 0.014	1.491 ± 0.024	**1.430 ± 0.023**	0.750 ± 0.014	0.823 ± 0.014
A3T-GCN(Zhu et al., 2020)	1.114 ± 0.008	1.119 ± 0.018	2.045 ± 0.021	2.008 ± 0.016	1.469 ± 0.027	1.475 ± 0.029	0.781 ± 0.011	0.872 ± 0.017
T-GCN (Zhao et al., 2019)	1.117 ± 0.011	1.111 ± 0.022	2.045 ± 0.027	2.008 ± 0.017	1.479 ± 0.012	1.481 ± 0.029	0.764 ± 0.011	0.846 ± 0.020
MPNN LSTM (Panagopoulos, Nikolentzos and Vazirgiannis, 2021)	1.116 ± 0.023	1.129 ± 0.021	2.053 ± 0.041	2.007 ± 0.010	1.485 ± 0.028	1.458 ± 0.013	0.795 ± 0.010	0.905 ± 0.017
AGCRN (Bai et al., 2020)	1.120 ± 0.010	1.116 ± 0.017	2.039 ± 0.022	2.010 ± 0.009	1.469 ± 0.030	1.465 ± 0.026	0.788 ± 0.011	0.832 ± 0.020

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

Chapter 2

The Shapley Value in Machine Learning

Joint work with Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Sebastian Nilsson, and Rik Sarkar

2.1 Introduction

Measuring importance and the attribution of various gains is a central problem in many practical aspects of machine learning such as explainability (Lundberg and Lee, 2017), feature selection (Cohen, Dror and Ruppin, 2007), data valuation (Ghorbani and Zou, 2019), ensemble pruning (Rozemberczki and Sarkar, 2021) and federated learning (Wang et al., 2020; Fan et al., 2021). For example, one might ask: What is the importance of a feature in the decisions of a machine learning model? How much is an individual data point worth? Which models are the most valuable in an ensemble? These questions have been addressed in different domains using specific approaches. Interestingly, there is also a general and unified approach to these questions as a solution to a *transferable utility* (TU) cooperative game. In contrast with other approaches, solution concepts of TU games are theoretically motivated with axiomatic properties. The best known solution is the *Shapley value* (Shapley, 1953) characterized by desiderata that include fairness, symmetry, and efficiency (Chalkiadakis, Elkind and Wooldridge, 2011).

DOI: 10.14754/CEU.2024.08

In the TU setting, a cooperative game consists of a *player set* and a scalar-valued *characteristic function* that defines the value of *coalitions* (subsets of players). In such a game, the Shapley value offers a rigorous and intuitive way to distribute the collective value (e.g. the revenue, profit, or cost) of the team across individuals. To apply this idea to machine learning, we need to define two components: the player set and the characteristic function. In a machine learning setting *players* may be represented by a set of input features, reinforcement learning agents, data points, models in an ensemble, or data silos. The characteristic function can then describe the goodness of fit for a model, reward in reinforcement learning, financial gain on instance level predictions, or out-of-sample model performance. We provide an example about model valuation in an ensemble (Rozemberczki and Sarkar, 2021) in Figure 2.1.

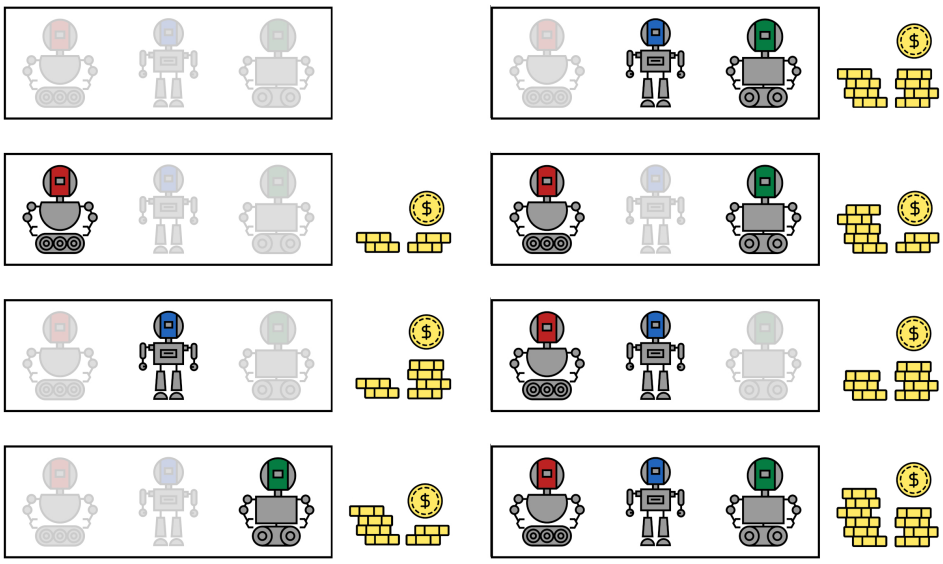


Figure 2.1: The Shapley value can be used to solve cooperative games. An ensemble game is a machine learning application for it – models in an ensemble are players (red, blue, and green robots) and the financial gain of the predictions is the payoff (coins) for each possible coalition (rectangles). The Shapley value can distribute the gain of the grand coalition (right bottom corner) among models.

We introduce basic definitions of cooperative games and present the Shapley value, a solution concept that can allocate gains in these games to individual players. We discuss its properties and emphasize why these are important in machine learning. We overview applications of the

Shapley value in machine learning: feature selection, data valuation, explainability, reinforcement learning, and model valuation. Finally, we discuss the limitations of the Shapley value and point out future directions. The paper is supported by a collection of related work under `https://github.com/AstraZeneca/awesome-shapley-value`.

2.2 Background

This section introduces cooperative games and the Shapley value followed by its properties. We also provide an illustrative running example for our definitions.

2.2.1 Cooperative Games and the Shapley Value

Definition 4 ***Player set and coalitions.*** *Let $\mathcal{N} = \{1, \ldots, n\}$ be the finite set of players. We call each non-empty subset $\mathcal{S} \subseteq \mathcal{N}$ a coalition and $\mathcal{N}$ itself the grand coalition.*

Definition 5 ***Cooperative game.*** *A TU game is defined by the pair $(\mathcal{N}, v)$ where $v : 2^{\mathcal{N}} \to \mathbb{R}$ is a mapping called the characteristic function or the coalition function of the game assigning a real number to each coalition and satisfying $v(\emptyset) = 0$.*

It is important to note that there are no additional requirements towards the coalition function. For example, it does not have to be increasing in set addition (the marginal contribution of a player to a coalition can be negative).

Example 1 *Let us consider a 3-player cooperative game where $\mathcal{N} = \{1, 2, 3\}$. The characteristic function defines the payoff for each coalition. Let these payoffs be given as:*

$$v(\emptyset) = 0; \quad v(\{1\}) = 7; \quad v(\{2\}) = 11; \quad v(\{3\}) = 14;$$
$$v(\{1,2\}) = 18; \quad v(\{1,3\}) = 21; \quad v(\{2,3\}) = 23; \quad v(\{1,2,3\}) = 25.$$

Definition 6 ***Set of feasible payoff vectors.*** *Let us define $\mathcal{Z}(\mathcal{N}, v) = \{\mathbf{z} \in \mathbb{R}^{\mathcal{N}} \mid \sum_{i \in \mathcal{N}} z_i \leq v(\mathcal{N})\}$ the set of feasible payoff vectors for the cooperative game $(\mathcal{N}, v)$.*

DOI: 10.14754/CEU.2024.08

Definition 7 ***Solution concept and solution vector.*** *Solution concept* Φ *is a mapping associating a subset* $\Phi(\mathcal{N}, v) \subseteq \mathcal{Z}(\mathcal{N}, v)$ *to every TU game* $(\mathcal{N}, v)$*. A solution vector* $\phi(\mathcal{N}, v) \in \mathbb{R}^{\mathcal{N}}$ *to the cooperative game* $(\mathcal{N}, v)$ *satisfies solution concept* Φ *if* $\phi(\mathcal{N}, v) \in \Phi(\mathcal{N}, v)$*. Solution concept* Φ *is single-valued if for every* $(\mathcal{N}, v)$ *the set* $\Phi(\mathcal{N}, v)$ *is a singleton.*

A solution concept defines an allocation principle through which rewards can be given to the individual players. The sum of these rewards cannot exceed the value of the grand coalition $v(\mathcal{N})$. Solution vectors are specific allocations satisfying the principles of the solution concept.

Definition 8 ***Permutations of the player set.*** *Let* $\Pi(\mathcal{N})$ *be the set of all permutations defined on* $\mathcal{N}$*, a specific permutation is written as* $\pi \in \Pi(\mathcal{N})$ *and* $\pi(i)$ *is the position of player* $i \in \mathcal{N}$ *in permutation* π*.*

Definition 9 ***Predecessor set.*** *Let the set of predecessors of player* $i \in \mathcal{N}$ *in permutation* π *be the coalition:*

$$\mathcal{P}_i^{\pi} = \{j \in \mathcal{N} \mid \pi(j) < \pi(i)\}.$$

Let us imagine that the permutation of the players in our illustrative game is $\pi = (3, 2, 1)$. Under this permutation the predecessor set of the 1^{st} player is $\mathcal{P}_1^{\pi} = \{3, 2\}$, that of the 2^{nd} player is $\mathcal{P}_2^{\pi} = \{3\}$ and $\mathcal{P}_3^{\pi} = \emptyset$.

Definition 10 ***Shapley value.*** *The Shapley value (Shapley, 1953) is a single-valued solution concept for cooperative games. The* i^{th} *component of the single solution vector satisfying this solution concept for any cooperative game* $(\mathcal{N}, v)$ *is given by Equation 2.1.*

$$\phi_i^{Sh} = \frac{1}{|\Pi(\mathcal{N})|} \sum_{\pi \in \Pi(\mathcal{N})} \underbrace{[v(\mathcal{P}_i^{\pi} \cup \{i\}) - v(\mathcal{P}_i^{\pi})]}_{\substack{\textbf{\textit{Player } i\textbf{\textit{'s marginal contribution}} \\ \textbf{\textit{in permutation }} \pi}} \tag{2.1}$$

The Shapley value of a player is the average marginal contribution of the player to the value of the predecessor set over every possible permutation of the player set. Table 2.1 contains manual

calculations of the players' marginal contributions to each permutation and their Shapley values in Example 1.

	Marginal Contribution		
Permutation	**Player 1**	**Player 2**	**Player 3**
(1, 2, 3)	7	11	7
(1, 3, 2)	7	4	14
(2, 1, 3)	7	11	7
(2, 3, 1)	2	11	12
(3, 1, 2)	7	4	14
(3, 2, 1)	2	9	14
Shapley value	32/6	50/6	68/6

Table 2.1: The permutations of the player set, marginal contributions of the players in each permutation and the Shapley values.

2.2.2 Properties of the Shapley Value

We define the solution concept properties that characterize the Shapley value and emphasize their relevance and meaning in a *feature selection game*. In this game input features are players, coalitions are subsets of features and the payoff is a scalar valued goodness of fit for a machine learning model using these inputs.

Definition 11 ***Null player.*** *Player i is called a null player if $v(\mathcal{S} \cup \{i\}) = v(\mathcal{S})$ $\forall \mathcal{S} \subseteq \mathcal{N} \setminus \{i\}$. A solution concept Φ satisfies the null player property if for every game $(\mathcal{N}, v)$, every $\phi(\mathcal{N}, v) \in \Phi(\mathcal{N}, v)$, and every null player i it holds that $\phi_i(\mathcal{N}, v) = 0$.*

In the feature selection game a solution concept with the null player property assigns zero value to those features that never increase the goodness of fit when added to the feature set.

Definition 12 ***Efficiency.*** *A solution concept Φ is efficient or Pareto optimal if for every game $(\mathcal{N}, v)$ and every solution vector $\phi(\mathcal{N}, v) \in \Phi(\mathcal{N}, v)$ it holds that $\sum_{i \in \mathcal{N}} \phi_i(\mathcal{N}, v) = v(\mathcal{N})$.*

Consider the goodness of fit of the model trained using the whole set of features. The importance measures assigned to features by an efficient solution concept sum to this goodness of fit. This

DOI: 10.14754/CEU.2024.08

allows for quantifying the contribution of features to the performance of a model trained on the whole feature set.

Definition 13 ***Symmetry.*** *Two players i and j are symmetric if $v(\mathcal{S} \cup \{i\}) = v(\mathcal{S} \cup \{j\})$ $\forall \mathcal{S} \subseteq \mathcal{N} \setminus \{i, j\}$. A solution concept Φ satisfies symmetry if for all $(\mathcal{N}, v)$ for all $\phi(\mathcal{N}, v) \in \Phi(\mathcal{N}, v)$ and all symmetric players $i, j \in \mathcal{N}$ it holds that $\phi_i(\mathcal{N}, v) = \phi(\mathcal{N}, v)_j$.*

The symmetry property implies that if two features have the same marginal contribution to the goodness of fit when added to any coalition then the importance of the two features is the same. This property is essentially a fair treatment of inputs and results in identical features receiving the same importance score.

Definition 14 ***Linearity.*** *A single-valued solution concept Φ satisfies linearity if for any two games $(\mathcal{N}, v)$ and $(\mathcal{N}, w)$, and for the solution vector of the TU game given by $(\mathcal{N}, v + w)$ it holds that*

$$\phi_i(\mathcal{N}, v + w) = \phi_i(\mathcal{N}, v) + \phi_i(\mathcal{N}, w), \quad \forall i \in \mathcal{N}.$$

Let us imagine a binary classifier and two sets of data points – on both of these datasets, we can define feature selection games with binary cross entropy-based payoffs. The Shapley values of input features in the feature selection game calculated on the pooled dataset would be the same as adding together the Shapley values calculated from the two datasets separately.

These four properties together characterize the Shapley value.

Theorem 1 (Shapley, 1953) *A single-valued solution concept satisfies the null player, efficiency, symmetry, and linearity properties if and only if it is the Shapley value.*

2.3 Approximations of the Shapley Value

Shapley value computation requires a factorial number of characteristic function evaluations, resulting in factorial time complexity. This is prohibitive in a machine learning context when

DOI: 10.14754/CEU.2024.08

each evaluation can correspond to training a machine learning model. For this reason, machine learning applications use a variety of Shapley value approximation methods we discuss in this section. In the following discussion $\widehat{\phi}_i^{Sh}$ denotes an approximated Shapley value for player $i \in \mathcal{N}$.

2.3.1 Monte Carlo Permutation Sampling

Monte Carlo permutation sampling for the general class of cooperative games was first proposed by Castro, Gómez and Tejada (2009) to approximate the Shapley value in linear time. Their method performs a sampling-based approximation, at each iteration, a random element from the permutations of the player set is drawn. The marginal contributions of the players in the sampled permutation are scaled down by the number of samples (which is equivalent to taking an average) and added to the approximated Shapley values from the previous iteration. Castro, Gómez and Tejada (2009) provide asymptotic error bounds for this approximation algorithm via the central limit theorem when the variance of the marginal contributions is known. Maleki et al. (2013) extended the analysis of this sampling approach by providing error bounds when either the variance or the range of the marginal contributions is known via Chebyshev's and Hoeffding's inequalities. Their bounds hold for a finite number of samples in contrast to the previous asymptotic bounds.

Stratified Sampling for Variance Reduction

In addition to extending the analysis of Monte Carlo estimation, Maleki et al. (2013) demonstrate how to improve Shapley value approximation when sampling can be *stratified* by dividing the permutations of the player set into homogeneous, non-overlapping sub-populations. In particular, they show that if the set of permutations can be grouped into strata with similar marginal gains for players, then the approximation will be more precise. Following this, Castro, Gómez et al. (2017) explored stratified sampling approaches using strata defined by the set of all

DOI: 10.14754/CEU.2024.08

marginal contributions when the player is in a specific position within the coalition. Burgess and Chapman (2021) propose stratified sampling approaches designed to minimize the uncertainty of the estimate via a stratified empirical Bernstein bound.

Other Variance Reduction Techniques

Following the stratified approaches of Maleki et al. (2013); Castro, Gómez et al. (2017); Burgess and Chapman (2021), Illés and Kerényi (2019) propose an alternative variance reduction technique for the sample mean. Instead of generating a random sequence of samples, they instead generate a sequence of ergodic but not independent samples, taking advantage of negative correlation to reduce the sample variance. Mitchell et al. (2021) show that other Monte Carlo variance reduction techniques can also be applied to this problem, such as antithetic sampling (Lomeli et al., 2019; Rubinstein and Kroese, 2016). A simple form of antithetic sampling uses both a randomly sampled permutation and its reverse. Finally, Touati, Radjef and Lakhdar (2021) introduce a Bayesian Monte Carlo approach to Shapley value calculation, showing that Shapley value estimation can be improved by using Bayesian methods.

2.3.2 Multilinear Extension

By inducing a probability distribution over the subsets $\mathcal{S}$ where $\mathcal{E}_i$ is a random subset that does not include player i and each player is included in a subset with probability q, Owen (1972) demonstrated that the sum over subsets in Definition 10 can also be represented as an integral $\int_0^1 e_i(q)dq$ where $e_i(q) = \mathbb{E}[v(\mathcal{E}_i \cup i) - v(\mathcal{E}_i)]$. Sampling over q therefore provides an approximation method – the multilinear extension. For example, Mitchell et al. (2021) uses the trapezoid rule to sample q at fixed intervals while Okhrati and Lipani (2021) proposes incorporating antithetic sampling as a variance reduction technique.

DOI: 10.14754/CEU.2024.08

2.3.3 Linear Regression Approximation

In their seminal work Lundberg and Lee (2017) apply Shapley values to feature importance and explainability (SHAP values), demonstrating that Shapley values for TU games can be approximated by solving a weighted least squares optimization problem. Their main insight is the computation of Shapley values by approximately solving the following optimization problem:

$$w_{\mathcal{S}} = \frac{|\mathcal{N}| - 1}{\binom{|\mathcal{N}|}{|\mathcal{S}|}|\mathcal{S}|(|\mathcal{N}| - |\mathcal{S}|)} \tag{2.2}$$

$$\underset{\widehat{\phi}_0^{Sh},\ldots,\widehat{\phi}_n^{Sh}}{min} \sum_{\mathcal{S}\subseteq\mathcal{N}} w_{\mathcal{S}} \left(\widehat{\phi}_0^{Sh} + \sum_{i\in\mathcal{S}} \widehat{\phi}_i^{Sh} - v(\mathcal{S}) \right) \tag{2.3}$$

$$s.t.\ \ \widehat{\phi}_0^{Sh} = v(\emptyset), \qquad \widehat{\phi}_0^{Sh} + \sum_{i\in\mathcal{N}} \widehat{\phi}_i^{Sh} = v(\mathcal{N}). \tag{2.4}$$

The definition of weights in Equation (2.2) and the objective function in Equation (2.3) implies the evaluation of $v(\cdot)$ for 2^n coalitions. To address this Lundberg and Lee (2017) propose approximating this by subsampling the coalitions. Note that $w_{\mathcal{S}}$ is higher when coalitions are large or small. Covert and Lee (2021) extend the study of this method, finding that while SHAP is a consistent estimator, it is not an unbiased one. By proposing and analyzing a variation of this unbiased method, they conclude that while there is a small bias incurred by SHAP it has a significantly lower variance than the corresponding unbiased estimator. Covert and Lee (2021) then propose a variance reduction method for SHAP, improving convergence speed by a magnitude through sampling coalitions in pairs with each selected alongside its complement.

DOI: 10.14754/CEU.2024.08

2.4 Machine Learning and the Shapley Value

Our discussion about applications of the Shapley value in machine learning focuses on the formulation of the cooperative games, definition of the player set and payoffs, Shapley value approximation, and the time complexity of the approximation. We summarized the most important application areas in Table 2.2 and grouped the relevant works by the problem solved.

2.4.1 Feature Selection

The feature selection game treats input features of a machine learning model as players and model performance as the payoff (Guyon and Elisseeff, 2003; Fryer, Strümke and Nguyen, 2021). The Shapley values of features quantify how much individual features contribute to the model's performance on a set of data points.

Definition 15 ***Feature selection game.*** *Let the player set be $\mathcal{N} = \{1, \ldots, n\}$, for $\mathcal{S} \subseteq \mathcal{N}$ the train and test feature vector sets are $\mathcal{X}_{\mathcal{S}}^{Train} = \left\{\boldsymbol{x}_i^{Train} | i \in \mathcal{S}\right\}$ and $\mathcal{X}_{\mathcal{S}}^{Test} = \left\{\boldsymbol{x}_i^{Test} | i \in \mathcal{S}\right\}$. Let $f_{\mathcal{S}}(\cdot)$ be a machine learning model trained using $\mathcal{X}_{\mathcal{S}}^{Train}$ as input, then the payoff is $v(\mathcal{S}) = g(\boldsymbol{y}, \widehat{\boldsymbol{y}}_{\mathcal{S}})$ where $g(\cdot)$ is a goodness of fit function, $\boldsymbol{y}$ and $\widehat{\boldsymbol{y}}_{\mathcal{S}} = f_{\mathcal{S}}(\mathcal{X}_{\mathcal{S}}^{Test})$ are the ground truth and predicted targets.*

Shapley values, and close relatives such as the Banzhaf index (Banzhaf III, 1964), have been studied as a measure of feature importance in various contexts (Cohen, Dror and Ruppin, 2007; Pintér, 2011; Sun et al., 2012; Williamson and Feng, 2020; Tripathi, Hemachandra and Trivedi, 2020). Using these importance estimates, features can be ranked and selected or removed accordingly. This approach has been applied to various tasks such as vocabulary selection in natural language processing (Patel et al., 2021) and feature selection in human action recognition (Guha, Khan et al., 2021).

2.4.2 Data Valuation

In the data valuation game training set data points are players and the payoff is defined by the goodness of fit achieved by a model on the test data. Computing the Shapley value of players in a data valuation game measures how much data points contribute to the performance of the model.

Definition 16 ***Data valuation game.*** *Let the player set be* $\mathcal{N} = \{(\boldsymbol{x}_i, y_i) \mid 1 \leq i \leq n\}$ *where* $\boldsymbol{x}_i$ *is the input feature vector and* y_i *is the target. Given the coalition* $\mathcal{S} \subseteq \mathcal{N}$ *let* $f_{\mathcal{S}}(\cdot)$ *be a machine learning model trained on* $\mathcal{S}$*. Let us denote the test set feature vectors and targets as* $\mathcal{X}$ *and* $\mathcal{Y}$*, given* $f_{\mathcal{S}}(\cdot)$ *the set of predicted labels is defined as* $\widehat{\mathcal{Y}} = \{f_{\mathcal{S}}(\boldsymbol{x}) | \boldsymbol{x} \in \mathcal{X}\}$*. Then the payoff of a model trained on the data points* $\mathcal{S} \subseteq \mathcal{N}$ *is* $v(\mathcal{S}) = g(\mathcal{Y}, \widehat{\mathcal{Y}})$ *where* $g(\cdot)$ *is a goodness of fit metric.*

The Shapley value is not the only method for data valuation – earlier works used function utilization (Koh and Liang, 2017), leave-one-out testing (Cook, 1977) and core sets (Dasgupta et al., 2009). However, these methods fall short (Jia et al., 2019; Ghorbani and Zou, 2019; Kwon and Zou, 2021) when there are fairness requirements from the data valuation technique. Ghorbani and Zou (2019) proposed a framework of utilizing Shapley value in a data-sharing system; Jia et al. (2019) advanced this work with more efficient algorithms to approximate the Shapley value for data valuation. The distributional Shapley value has been discussed by Ghorbani, Kim and Zou (2020) who argued that maintaining privacy is hard during Shapley value computation. Their method calculates the Shapley value over a distribution which solves problems such as lack of privacy. The computation time of this can be reduced as Kwon, Rivas and Zou (2021) point out with approximation methods optimized for specific models.

2.4.3 Federated Learning

A federated learning scenario can be seen as a cooperative game by modeling the data owners as players who cooperate to train a high-quality machine learning model (Liu et al., 2021*b*).

Definition 17 ***Federated learning game.*** *In this game players are a set of labeled dataset owners* $\mathcal{N} = \{(\mathcal{X}_i, \mathcal{Y}_i) | 1 \leq i \leq n\}$ *where* $\mathcal{X}_i$ *and* $\mathcal{Y}_i$ *are the feature and label sets owned by the* i^{th} *silo. Let* $(\mathcal{X}, \mathcal{Y})$ *be a labeled test set,* $\mathcal{S} \subseteq \mathcal{N}$ *a coalition of data silos,* $f_{\mathcal{S}}(\cdot)$ *a machine learning model trained on* $\mathcal{S}$*, and* $\widehat{\mathcal{Y}}_{\mathcal{S}}$ *the labels predicted by* $f_{\mathcal{S}}(\cdot)$ *on* $\mathcal{X}$*. The payoff of* $\mathcal{S} \subseteq \mathcal{N}$ *is* $v(\mathcal{S}) = g(\mathcal{Y}, \widehat{\mathcal{Y}}_{\mathcal{S}})$ *where* $g(\cdot)$ *is a goodness of fit metric.*

The system described by Liu et al. (2021*b*) uses Monte Carlo sampling to approximate the Shapley value of the data silos. Given the potentially overlapping nature of the datasets, the use of configuration games, an extension of the Shapley value, could be an interesting future direction for federated learning.

2.4.4 Explainable Machine Learning

In explainable machine learning the Shapley value is used to measure the contributions of input features to the output of a machine learning model at the instance level. Given a specific data point, the goal is to decompose the model prediction and assign Shapley values to individual features of the instance. There are universal solutions to this challenge that are model agnostic and designs customized for deep learning (Chen et al., 2018*a*; Ancona, Oztireli and Gross, 2019), classification trees (Lundberg and Lee, 2017), and graphical models (Liu et al., 2020; Singal, Michailidis and Ng, 2021).

Universal Explainability

A cooperative game for universal explainability is completely model agnostic; the only requirement is that a scalar-valued output can be generated by the model such as the probability of a class label being assigned to an instance.

Definition 18 ***Universal explainability game.*** *Let us denote the machine learning model of interest with* $f(\cdot)$ *and let the player set be the feature values of a single data instance:* $\mathcal{N} =$

$\{x_i | 1 \leq i \leq n\}$. *The payoff of a coalition* $\mathcal{S} \subseteq \mathcal{N}$ *in this game is the scalar-valued prediction* $v(\mathcal{S}) = \widehat{y}_{\mathcal{S}} = f(\mathcal{S})$ *calculated from the subset of feature values.*

Calculating the Shapley value in a game like this offers a complete decomposition of the prediction because the *efficiency* axiom holds. The Shapley values of feature values are explanatory attributions to the input features and missing input feature values are imputed with a reference value such as the mean computed from multiple instances (Lundberg and Lee, 2017; Covert and Lee, 2021). The pioneering Shapley value-based universal explanation method SHAP (Lundberg and Lee, 2017) proposes a linear time approximation of the Shapley values which we discussed in Section 2.3. This approximation has shortcomings and implicit assumptions about the features which are addressed by newer Shapley value-based explanation techniques. For example, in Frye et al. (2020) the input features are not necessarily independent, Frye, Rowat and Feige (2020) restricts the permutations based on known causal relationships, and in Covert and Lee (2021) the proposed technique improves the convergence guarantees of the approximation. Several methods generalize SHAP beyond feature values to give attributions to first-order feature interactions (Sundararajan and Najmi, 2020; Sundararajan, Dhamdhere and Agarwal, 2020). However, this requires that the player set is redefined to include feature interaction values.

Deep Learning

In neuron explainability games neurons are players and attributions to the neurons are payoffs. The primary goal of Shapley value-based explanations in deep learning is to solve these games and compute attributions to individual neurons and filters (Ghorbani and Zou, 2020; Ancona, Oztireli and Gross, 2019).

Definition 19 ***Neuron explainability game.*** *Let us consider* $f_{IN}(\cdot)$ *the encoder layer of a neural network and* $\boldsymbol{x}$ *the input feature vector to the encoder. In the neuron explainability game the player set is* $\mathcal{N} = f_{IN}(\boldsymbol{x}) = \{h_1, \ldots, h_n\}$ *- each player corresponds to the output of a neuron in*

the final layer of the encoder. The payoff of coalition $\mathcal{S} \subseteq \mathcal{N}$ is defined as the predicted output $v(\mathcal{S}) = \widehat{y}_{\mathcal{S}} = f_{OUT}(\mathcal{S})$ where $f_{OUT}(\cdot)$ is the head layer of the neural network.

In practical terms, the payoffs are the output of the neural network obtained by masking out certain neurons. Using the Shapley values obtained in these games the value of individual neurons can be quantified. At the same time, some deep learning specific Shapley value-based explanation techniques have designs and goals that are aligned with the games described in universal explainability. These methods exploit the structure of the input data (Chen et al., 2018*a*) or the nature of feature interactions (Zhang et al., 2021) to provide efficient computations of attributions.

Graphical Models

Compared to universal explanations the graphical model-specific techniques restrict the admissible set of player set permutations considered in the attribution process. These restrictions are defined based on known causal relations and permutations are generated by various search strategies on the graph describing the probabilistic model (Heskes et al., 2020; Liu et al., 2020; Singal, Michailidis and Ng, 2021). Methods are differentiated from each other by how restrictions are defined and how permutations are restricted.

Relational Machine Learning

In the relational machine learning domain the Shapley value is used to create edge importance attributions of instance-level explanations (Duval and Malliaros, 2021; Yuan et al., 2021). Essentially the Shapley value in these games measures the average marginal change in the outcome variable as one adds a specific edge to the edge set in all of the possible edges set permutations. It is worth noting that the edge explanation and attribution techniques proposed could be generalized to provide node attributions.

DOI: 10.14754/CEU.2024.08

Definition 20 ***Relational explainability game.*** *Let us define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{N})$ where $\mathcal{V}$ and $\mathcal{N}$ are the vertex and edge sets. Given the relational machine learning model $f(\cdot)$, node feature matrix $\boldsymbol{X}$, node $u \in \mathcal{V}$, the payoff of coalition $\mathcal{S} \subseteq \mathcal{V}$ in the graph machine learning explanation game is defined as the node level prediction $v(\mathcal{S}) = \widehat{y}_{\mathcal{S},u} = f(\boldsymbol{X}, \mathcal{V}, \mathcal{S}, u)$.*

2.4.5 Multi-Agent Reinforcement Learning

Global reward multi-agent reinforcement learning problems can be modeled as TU games (Wang et al., 2021; Li et al., 2021) by defining the player set as the set of agents and the payoff of coalitions as a global reward. The Shapley value allows an axiomatic decomposition of the global reward achieved by the agents and the fair attribution of credit assignments to agents.

2.4.6 Model Valuation in Ensembles

The Shapley value can be used to assess the contributions of machine learning models to a composite model in ensemble games. In these games, players are models in an ensemble and payoffs are decided by whether an instance level prediction made by the model is correct.

Definition 21 ***Ensemble game***. *Let us consider a single target - feature instance denoted by $(y, \boldsymbol{x})$. The player set in ensemble games is defined by a set of machine learning models $\mathcal{N} = \{f_i(\cdot) | 1 \leq i \leq n\}$ that operate on the feature set. The predicted target output by the ensemble $\mathcal{S} \subseteq \mathcal{N}$ is defined as $\widehat{y}_{\mathcal{S}} = \tilde{f}(\mathcal{S}, \boldsymbol{x})$ where $\tilde{f}(\cdot)$ is a prediction aggregation function. The payoff of $\mathcal{S}$ is $v(\mathcal{S}) = g(y, \widehat{y}_{\mathcal{S}})$ where $g(\cdot)$ is a goodness of fit metric.*

The ensemble games described by Rozemberczki and Sarkar (2021) are formulated as a special subclass of voting games. This allows the use of precise game-specific approximation (Fatima, Wooldridge and Jennings, 2008) techniques and because of this the Shapley value estimates are obtained in quadratic time and have a tight approximation error. The games themselves are model agnostic concerning the player set – ensembles can be formed by heterogeneous types of machine learning models that operate on the same inputs.

DOI: 10.14754/CEU.2024.08

2.5 Discussion

The Shapley value has a wide-reaching impact in machine learning, but it has limitations and certain extensions of the Shapley value could have important applications in machine learning.

2.5.1 Limitations

Computation Time

Computing the Shapley value for each player naively in a TU game takes factorial time. In some machine learning application areas such as multi-agent reinforcement learning and federated learning where the number of players is small, this is not an issue. However, in large scale data valuation (Kwon, Rivas and Zou, 2021; Kwon and Zou, 2021), explainability (Lundberg and Lee, 2017), and feature selection (Patel et al., 2021) settings the exact calculation of the Shapley value is not tractable. In Sections 2.3 and 2.4 we discussed approximation techniques proposed to make Shapley value computation possible. In some cases, asymptotic properties of these Shapley value approximation techniques are not well understood – see for example Chen et al. (2018*a*).

Interpretability

By definition, the Shapley values are the average marginal contributions of players to the payoff of the grand coalition computed from all permutations (Shapley, 1953). Theoretical interpretations like this one are not intuitive and not useful for non-game theory experts. This means that translating the meaning of Shapley values obtained in many application areas to actions is troublesome (Kumar et al., 2020). For example in a data valuation scenario: is a data point with a twice as large Shapley value as another one twice as valuable? Answering a question like this requires a definition of a cooperative game that is intuitive.

DOI: 10.14754/CEU.2024.08

Axioms Do Not Hold Under Approximations

As we discussed most applications of the Shapley value in machine learning use approximations. The fact that under these approximations the desired axiomatic properties of the Shapley value do not hold is often overlooked (Sundararajan and Najmi, 2020). This is problematic because most works argue for the use of Shapley value based on these axioms. In our view, this is the greatest unresolved issue in the application of the Shapley value.

2.5.2 Future Research Directions

Hierarchy of the Coalition Structure

The Shapley value has a constrained version called Owen value (Owen, 1977) in which only permutations satisfying conditions defined by a *coalition structure* - a partition of the player set - are considered. The calculation of the Owen value is identical to that of the Shapley value, with the exception that only those permutations are taken into account where the players in any of the subsets of the *coalition structure* follow each other. In several real-world data and feature valuation scenarios even more complex hierarchies of the coalition, the structure could be useful. Having a nested hierarchy imposes restrictions on the admissible permutations of the players and changes player valuation. Games with such nested hierarchies are called level structure games in game theory. Winter (1989) presents the Winter value a solution concept to level structure games - such games are yet to receive attention in the machine learning literature.

Overlapping Coalition Structure

Traditionally, it is assumed that players in a coalition structure are allocated in disjoint partitions of the grand coalition. Allowing players to belong to overlapping coalitions in configuration games (Albizuri, Aurrecoechea et al., 2006) could have several applications in machine learning. For example in a data-sharing - feature selection scenario multiple data owners might have access to the same features - a feature can belong to overlapping coalitions.

DOI: 10.14754/CEU.2024.08

Solution Concepts Beyond the Shapley Value

The Shapley value is a specific solution concept of cooperative game theory with intuitive axiomatic properties (Section 2.2). At the same time it has limitations with respect to computation constraints and interpretability (Sections 2.3 and 2.5). Cooperative game theory offers other solution concepts such as the *core*, *nucleolus*, *stable set*, and *kernel* with their own axiomatizations. For example, the *core* has been used for model explainability and feature selection (Yan and Procaccia, 2021). Research into the potential applications of these solution concepts is lacking.

2.6 Conclusion

In this paper we discussed the Shapley value, examined its axiomatic characterizations and frequently used Shapley value approximations. We defined and reviewed its uses in machine learning, highlighted issues with the Shapley value and potential new application and novel research areas in machine learning.

Application	Reference	Payoff	Approximation	Time
Feature Selection	Cohen, Dror and Ruppin (2007)	Validation loss	Exact	$O(\|\mathcal{N}\|!)$
	Sun et al. (2012)	Mutual information	Exact	$O(\|\mathcal{N}\|!)$
	Williamson and Feng (2020)	Validation loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Tripathi, Hemachandra and Trivedi (2020)	Training loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Patel et al. (2021)	Validation loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Guha, Khan et al. (2021)	Validation loss	Exact	$O(\|\mathcal{N}\|!)$
Data Valuation	Jia et al. (2019)	Validation loss	Restricted Monte Carlo sampling	$O(\sqrt{\|\mathcal{N}\|}\log\|\mathcal{N}\|^2)$
	Ghorbani and Zou (2019)	Validation loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Shim et al. (2021)	Validation loss	Exact	$O(\|\mathcal{N}\|\log\|\mathcal{N}\|)$
	Deutch et al. (2021)	Validation loss	Restricted Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Kwon, Rivas and Zou (2021)	Validation loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Kwon and Zou (2021)	Validation loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
Federated Learning	Liu et al. (2021*b*)	Validation loss	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
Universal Explainability	Lundberg and Lee (2017)	Attribution	Linear regression	$O(\|\mathcal{N}\|)$
	Sundararajan, Dhamdhere and Agarwal (2020)	Interaction attribution	Integrated gradients	$O(\|\mathcal{N}\|^2)$
	Sundararajan and Najmi (2020)	Interaction attribution	Integrated gradients	$O(\|\mathcal{N}\|^2)$
	Frye et al. (2020)	Attribution	Linear regression	$O(\|\mathcal{N}\|)$
	Frye, Rowat and Feige (2020)	Attribution	Linear regression	$O(\|\mathcal{N}\|)$
	Yuan et al. (2021)	Attribution	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Covert and Lee (2021)	Attribution	Linear regression	$O(\|\mathcal{N}\|)$
Explainability of Deep Learning	Chen et al. (2018*a*)	Attribution	Restricted Monte Carlo sampling	$O(2^{\|\mathcal{N}\|})$ or $O(\|\mathcal{N}\|)$
	Ancona, Oztireli and Gross (2019)	Neuron attribution	Voting game	$O(\|\mathcal{N}\|^2)$
	Ghorbani and Zou (2020)	Neuron attribution	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Zhang et al. (2021)	Interaction Attribution	Linear regression	$O(\|\mathcal{N}\|)$
Explainability of Graphical Models	Liu et al. (2020)	Attribution	Exact	$O(\|\mathcal{N}\|!)$
	Heskes et al. (2020)	Causal Attribution	Linear regression	$O(\|\mathcal{N}\|)$
	Wang, Wiens and Lundberg (2021)	Causal Attribution	Linear regression	$O(\|\mathcal{N}\|)$
	Singal, Michailidis and Ng (2021)	Causal Attribution	Linear regression	$O(\|\mathcal{N}\|)$
Explainability in Graph Machine Learning	Yuan et al. (2021)	Edge level attribution	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Duval and Malliaros (2021)	Edge level attribution	Linear regression	$O(\|\mathcal{N}\|)$
Multi-agent Reinforcement Learning	Wang et al. (2021)	Global reward	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
	Li et al. (2021)	Global reward	Monte Carlo sampling	$O(\|\mathcal{N}\|)$
Model Valuation in Ensembles	Rozemberczki and Sarkar (2021)	Predictive performance	Voting game	$O(\|\mathcal{N}\|^2)$

Table 2.2: An application area, payoff definition, Shapley value approximation technique, and computation time (the player set is noted by $\mathcal{N}$) based comparison of research works. Specific applications of the Shapley value are grouped together and ordered chronologically.

DOI: 10.14754/CEU.2024.08

DOI: 10.14754/CEU.2024.08

Chapter 3

Peer Effects in Directed Multiplex Networks

3.1 Introduction

Networks characterize a multitude of economic and social interactions. For instance, existing social relationships can shape how individuals create new connections and build social capital (Coleman, 1988), while ownership and management networks may influence the way companies operate within a competitive business landscape (Davis, Yoo and Baker, 2003; Gulati and Gargiulo, 1999). In international trade, geographic networks influence the direction and magnitude of exports and imports between countries (Helpman, Melitz and Rubinstein, 2008). Whenever different types of connections such as social or geographical have an impact on economic outcomes, it's crucial to take into account the observable network dependencies. Existing literature has analyzed, among other topics, peer effects (Sacerdote, 2001), social segregation (Lazarsfeld and Merton, 1954), production networks (Acemoglu et al., 2012), and migration networks (Ortega and Peri, 2013).

Connections across agents also have important policy implications. An intervention targeting certain individuals can have spillover effects, thus indirectly affecting the outcomes of other

DOI: 10.14754/CEU.2024.08

agents as well (Galeotti, Golub and Goyal, 2020). These spillover effects, also called peer effects, have therefore been of interest in multiple domains. For example, Centola (2010) studied the spread of behavior using randomized online communities, Ammermueller and Pischke (2009) estimated peer effects in primary school performance, Gwozdz et al. (2015) investigated peer effects in adolescent obesity while Maxwell (2002) analyzed the effect of peers on risky behaviors, among others.

Estimating peer effects is, however, hindered by multiple practical challenges. Jackson, Rogers and Zenou (2017) outline the difficulties in relation to analyzing networks. These are network endogeneity and homophily (i.e.: network formation is not random, individuals often connect with like-minded people), measurement errors in links between agents, misspecification of the model describing peers' interactions, and having multiple types of relationships. This paper contributes to the discussion on the last issue by proposing a model that relies on multiplex networks - networks where among the same set of agents multiple types of relationships are observed.

This work proposes a tractable model with peer effect parameters that can be estimated using traditional tools in spatial econometrics. While this model does not address every issue discussed by Jackson, Rogers and Zenou (2017), it provides an insight into how failing to take into account multiple, correlated networks can result in biased peer effect estimates. The findings of this paper imply that research on peer effects in any setting should consider potentially correlated underlying networks and control for them during the estimation of these effects.

The remainder of this paper is structured as follows. Section 3.2 introduces the theoretical model and connects it to the reduced form spatial autoregressive model that is used for the parameter estimations. Section 3.3 presents the experimental setup and discusses the results of Monte Carlo simulations using synthetic data. Section 3.4 provides an empirical example using data from Ferrali et al. (2020), illustrating that alternative empirical approaches yield significantly different, likely biased results. Section 3.5 concludes by summarizing the most important findings of this work and further research directions.

DOI: 10.14754/CEU.2024.08

3.2 Model

This section presents a quadratic utility model that captures peer effects both in a single network and in a multiplex network setting. Section 3.2.1 introduces the general framework using a single network, while Section 3.2.2 extends the model to multiplex networks.

3.2.1 Quadratic utility model with a single network

Consider a game of N players, where each player i has to decide their effort level $y_i \in \mathcal{R}_0^+$. In a social network setting, for example, y_i may represent decision variables such as time spent in a specific social setting or using a social media site. The action profile of the players, $\mathbf{y}$ is defined as:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix}$$

Each player's payoff is given as a function of the action profile $\mathbf{y}$ and an underlying network $\mathcal{G}^N$. $\mathcal{G}^N$ can be described by adjacency matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{1N} \\ \vdots & & \vdots & & \vdots \\ a_{N1} & \dots & a_{Nj} & \dots & a_{NN} \end{bmatrix}$$

The a_{ij} element of adjacency matrix $\mathbf{A}$ can be either (i) a binary value describing whether there is a network connection from player i to player j or (ii) a non-negative real number describing the strength of the network connection from player i to player j. In a quadratic utility

DOI: 10.14754/CEU.2024.08

model with peer effects, the payoff function of each player can be described by Equation (3.1).

$$\pi_i(\mathbf{y}) = \alpha_i y_i - \frac{1}{2} y_i^2 + \lambda \sum_{j=1}^{N} a_{ij} y_i y_j \tag{3.1}$$

where λ is the peer effect parameter. The intuition behind Equation 3.1 is that the payoff of each agent depends (i) on one's own actions, which provides direct utility but also results in a quadratic cost $[y_i]$, (ii) individual factors represented by α_i and (iii) the actions of the agents they interact with. In this setting, the effect agent j's action has on agent i's utility is proportional to a_{ij}. Ballester, Calvó-Armengol and Zenou (2006) present a similar model of peer effects, however, their model is not identifiable in the way presented in this paper. In this game, the first order conditions in a Nash-equilibrium $\forall i$ satisfy:

$$y_i = \alpha_i + \lambda \sum_{j=1}^{N} a_{ij} y_j$$

Let us assume that α_i, the individual factor, can be described by using a linear model. If $\alpha = \mathbf{X}\beta + \varepsilon$ where

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_i \\ \vdots \\ \varepsilon_N \end{bmatrix}$$

is a vector of disturbances, then

$$\mathbf{y} = \lambda \mathbf{A}\mathbf{y} + \mathbf{X}\beta + \varepsilon$$

which is a classical spatial autoregressive model. Lee (2003) and Lee, Liu and Lin (2010) present a wide set of estimators that can efficiently estimate the structural parameters in such a model under different assumptions on the distribution of disturbances. This and variants of this model (extended by spatially lagged exogenous variables or spatial autoregressive disturbances) have

DOI: 10.14754/CEU.2024.08

been extensively applied to estimate peer effects in the literature (see for example Lin (2010), Lin (2015), Hsieh and van Kippersluis (2018), Fortin and Yazbeck (2015) or Hsieh and Lee (2016)). This model, however, does not account for multiple types of relationships and cannot be directly applied to multiplex networks. The next section presents an extension of this model that incorporates multiple types of relationships.

3.2.2 Quadratic utility model with multiplex networks

Consider a multiplex network of D dimensions $\mathcal{G}^{D\times N}$, that is, among the same agents we may observe D distinct types of edges. In a real-life setting, for example, we could observe friendships, family relationships, geographical proximities, and many other connections between the same set of individuals. It is a plausible assumption that we have heterogeneous peer effects across different types of connections. That is, one derives different utility from the actions of different types of peers. Such a scenario can be represented with a model where each player's payoff is:

$$\pi_i(\mathbf{y}) = \alpha_i y_i - \frac{1}{2}y_i^2 + \sum_{d=1}^{D}\lambda_d \sum_{j=1}^{N} a_{ij}^d y_i y_j$$

where λ_d is the peer effect parameter of network d. With this specification, the first order conditions in a Nash-equilibrium $\forall i$ satisfy:

$$y_i = \alpha_i + \sum_{d=1}^{D}\lambda_d \sum_{j=1}^{N} a_{ij}^d y_j$$

If we model the individual effects in a linear fashion, that is $\alpha = \mathbf{X}\beta + \varepsilon$ then

$$\mathbf{y} = \sum_{d=1}^{D}\lambda_d \mathbf{A^d y} + \mathbf{X}\beta + \varepsilon \tag{3.2}$$

DOI: 10.14754/CEU.2024.08

which is a spatial autoregressive model with multiple weight matrices. Equation (3.2) can be rearranged by defining

$$\mathbf{S} \equiv \mathbf{I_N} - \sum_{d=1}^{D} \lambda_d \mathbf{A^d}$$

Using this definition, we can rewrite Equation (3.2) as

$$\mathbf{y} = \mathbf{S}^{-1}\mathbf{X}\beta + \mathbf{S}^{-1}\varepsilon$$

Notice that the right-hand side only contains the exogenous node characteristics and the multiplex network adjacencies.

A potential issue with the identification of such a model is that most estimation approaches assume that $\varepsilon_i \sim i.i.d.(0, \sigma^2)$. In a social network setting this assumption is likely to be violated due to homophily. Since neighbors in a social network usually have similar characteristics, their error terms are likely correlated as well. To address this issue, we can additionally assume that the error term can be modeled with another spatial autoregressive term. That is, the model is:

$$\mathbf{y} = \sum_{d=1}^{D} \lambda_d \mathbf{A^d}\mathbf{y} + \mathbf{X}\beta + \mathbf{u}$$

where

$$\mathbf{u} \equiv \sum_{d=1}^{D} \rho_d \mathbf{A^d}\mathbf{u} + \varepsilon \tag{3.3}$$

Equations (3.2.2) and (3.3) describe a $SARAR(p, q)$ model with multiple weight matrices where $p = q = D$ and the spatial weight matrices are the adjacency matrices of the multiplex network. Let us define

$$\mathbf{S} \equiv \mathbf{I_N} - \sum_{d=1}^{D} \lambda_d \mathbf{A^d}$$

and

$$\mathbf{R} \equiv \mathbf{I_N} - \sum_{d=1}^{D} \rho_d \mathbf{A^d}$$

DOI: 10.14754/CEU.2024.08

Then the reduced form is

$$\mathbf{y} = \mathbf{S}^{-1}\mathbf{X}\beta + \mathbf{S}^{-1}\mathbf{R}^{-1}\varepsilon$$

It has been shown by Liu, Lee and Bollinger (2010), Badinger and Egger (2011) and Lee and Liu (2010) that the structural model parameters of interest (λ_d, ρ_d) in such a setting can be identified efficiently by iterative GMM estimators. These works, however, do not contain microfoundations for the empirical model and focus exclusively on econometric theory and the estimation of the model parameters. At the same time, they all mention potential applications of such a model: Liu, Lee and Bollinger (2010) and Lee and Liu (2010) cite regional, urban and public economics as potential application domains, while Badinger and Egger (2011) mention trade, foreign direct investment, migration and social interactions as a potential use case for their estimator. To our knowledge, the only application in the literature in the domain of this paper is done by König, Liu and Zenou (2019) estimate R& D spillover effects with three weight matrices. In their model, however, the same reduced form arises from an industrial organization model of multiple markets with substitutable goods.

3.3 Monte Carlo evidence

This section presents Monte Carlo evidence using synthetic data to illustrate both that i) omitting a network from the estimation results in biased estimates and that ii) simultaneous estimation of the peer effect parameters eliminates such bias in three different network scenarios. The first set of experiments evaluates the estimator using Erdos-Renyi random graphs. The second experiment repeats the first exercise using networks that were generated according to the Barabasi-Albert preferential attachment model (Albert and Barabási, 2002). The last subsection evaluates the model in a setting where the presence of network connections is correlated with exogenous node characteristics, that is, the networks present homophily.

DOI: 10.14754/CEU.2024.08

3.3.1 Erdos-Renyi random graphs

In the first illustration of omitted network bias we evaluate the estimator using $D = 2$ Erdos-Renyi random graphs. These networks contain a fixed number of vertices where each pair of vertices is connected by an edge with independent probability p.

We use the following parameters across all experiments:

- The peer effect parameter of network 1 is $\lambda_1 = 0.3$.
- The peer effect parameter of network 2 is $\lambda_1 = 0.1$.
- There is a single covariate drawn from $X \sim U[0, 5]$. Its coefficient is $\beta = 1$.
- ε is drawn from a standard normal distribution, $\rho_d = 0 \, \forall d$.
- Each Monte Carlo simulation is repeated 500 times.

We test the behavior of the estimator for multiple network densities and network sizes. Each experiment is repeated using networks with 25, 50, and 100 nodes. We run simulations for all combinations of $p_1 \in \{0.1, 0.2\}$ and $p_2 \in \{0.05, 0.1, 0.15, 0.2\}$.

All parameter constellations above are evaluated in three settings:

- Running the estimation with both networks included.
- Estimating with using only one of the networks (2 cases).

The first setup shows that by including both networks the estimator can identify the peer effect parameters. The second and third cases illustrate that omitting a network from the estimation yields biased peer effect estimates for the network included in the estimation.

We evaluate every simulation using three measures. We calculate the bias of the estimator the standard way. That is, for each parameter θ we calculate

$$\text{BIAS} = \frac{\sum_{i=1}^{500} \hat{\theta}_i}{500} - \theta$$

DOI: 10.14754/CEU.2024.08

We also include two measures capturing the variance of the estimator. We calculate the mean absolute error (MAE) to show the average magnitude of the prediction error and the root of the mean squared error (RMSE) which, contrary to MAE, is sensitive to outliers. These measures are defined for each parameter θ as:

$$\text{MAE} = \frac{\sum_{i=1}^{500} |\hat{\theta}_i - \theta|}{500}$$

and

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{500} (\hat{\theta}_i - \theta)^2}{500}}$$

Table 3.1 presents the results of the first test case, where both networks are included in the estimation simultaneously. Bias in all test cases is relatively low. In the case of the peer effect parameters, it does not exceed 10% of the estimated true parameter value with only one exception (the relative bias in case of λ_2 in the last test case and 50 nodes is 17.9%). These results are consistent with the findings of Badinger and Egger (2011). On similar sample sizes, they find that the average relative bias of the estimator is approximately 8 percent. It holds in almost all test cases that increasing the number of nodes decreases the magnitude of the bias for the estimated peer effects.

Another prevalent pattern in the results shows that the size of the bias increases in the densities of the networks. This pattern can intuitively be due to the fact that higher network densities imply more overlap between the networks, and therefore it is more difficult to distinguish between the effects propagating through each individual network precisely.

The parameter of the exogenous covariate is estimated precisely in all cases. Its bias does not exceed 1.4% of the true parameter value in any of the test cases. These findings are, in general, consistent with previous literature evaluating the behavior of the proposed estimator (Badinger and Egger, 2011).

DOI: 10.14754/CEU.2024.08

Table 3.1: Monte Carlo simulation results of the regression including both networks simultaneously

Test case			**25 nodes**			**50 nodes**			**100 nodes**		
p network 1	p network 2	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	-0.0015	0.1042	0.1332	-0.0054	0.0789	0.1006	-0.0009	0.0548	0.0692
0.1	0.05	λ_1	-0.0041	0.0811	0.1064	0.0014	0.0834	0.1063	-0.0046	0.0983	0.1229
		λ_2	-0.0035	0.0767	0.0992	-0.0018	0.0714	0.0902	0.0042	0.0919	0.1164
		β	-0.0027	0.1119	0.1406	-0.0083	0.0825	0.1036	-0.002	0.0553	0.0696
0.1	0.1	λ_1	-0.0062	0.0896	0.1141	-0.0017	0.1139	0.1427	-0.0018	0.1323	0.1644
		λ_2	0.0028	0.0879	0.1129	0.0036	0.1077	0.1364	0.0021	0.1288	0.1599
		β	-0.0039	0.1149	0.1448	-0.008	0.0808	0.1026	-0.0017	0.0552	0.0698
0.1	0.15	λ_1	-0.0103	0.0996	0.1268	0.0035	0.1294	0.1636	0.0009	0.1455	0.1787
		λ_2	0.0075	0.1112	0.139	-0.0015	0.1261	0.16	-0.0009	0.1463	0.1771
		β	-0.0038	0.1167	0.147	-0.0084	0.0807	0.1022	-0.0016	0.0553	0.0698
0.1	0.2	λ_1	-0.0125	0.1022	0.1296	0.0092	0.1372	0.1746	0.0004	0.1536	0.1889
		λ_2	0.0099	0.1185	0.1484	-0.0072	0.1337	0.1695	-0.0004	0.1537	0.1883
		β	-0.0074	0.1122	0.1411	-0.0049	0.0787	0.1004	-0.0003	0.0553	0.07
0.2	0.05	λ_1	-0.0005	0.0932	0.1204	0.0029	0.0914	0.1174	-0.0053	0.108	0.1345
		λ_2	-0.0024	0.0777	0.1007	-0.0036	0.0782	0.1003	0.0046	0.1003	0.1258
		β	-0.0093	0.1164	0.1453	-0.008	0.0816	0.1027	-0.0016	0.0556	0.07
0.2	0.1	λ_1	0.0026	0.12	0.1531	-0.0001	0.1423	0.1788	-0.0006	0.1528	0.1975
		λ_2	-0.0023	0.1054	0.1377	0.0017	0.1354	0.1718	0.0008	0.1477	0.1898
		β	-0.0136	0.1188	0.1497	-0.0082	0.0793	0.1013	-0.0018	0.0559	0.0706
0.2	0.15	λ_1	-0.0031	0.1597	0.2099	0.009	0.171	0.2189	0.0008	0.1802	0.2283
		λ_2	0.007	0.1597	0.2054	-0.007	0.1671	0.2149	-0.0005	0.1793	0.2247
		β	-0.0123	0.1205	0.15	-0.0078	0.0792	0.1013	-0.0019	0.0558	0.0704
0.2	0.2	λ_1	-0.004	0.1734	0.2243	0.0195	0.1869	0.2365	-0.0019	0.1968	0.2482
		λ_2	0.0075	0.1739	0.2239	-0.0179	0.1836	0.2311	0.0021	0.1942	0.2459

Table 3.2: Monte Carlo simulation results of the biased regression including only Network 1

Test case			**25 nodes**			**50 nodes**			**100 nodes**		
p network 1	p network 2	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	0.0262	0.1106	0.1387	0.0149	0.0822	0.1051	0.0087	0.0567	0.0717
0.1	0.05	λ_1	0.0491	0.0894	0.1114	0.0766	0.0889	0.1064	0.0922	0.0936	0.1039
		β	0.0355	0.1102	0.1398	0.0135	0.0801	0.1033	0.008	0.0559	0.0709
0.1	0.1	λ_1	0.0611	0.0919	0.1136	0.0849	0.0938	0.1111	0.093	0.0943	0.1044
		β	0.0377	0.1099	0.1385	0.0135	0.0812	0.1037	0.008	0.0558	0.0705
0.1	0.15	λ_1	0.0663	0.094	0.1153	0.0852	0.0944	0.1115	0.093	0.0942	0.1044
		β	0.0382	0.1101	0.14	0.0131	0.0809	0.1033	0.0075	0.056	0.0708
0.1	0.2	λ_1	0.0667	0.0945	0.1155	0.0857	0.0944	0.1115	0.0934	0.0946	0.1047
		β	0.0041	0.1188	0.1501	0.0066	0.0795	0.1013	0.0064	0.0553	0.0699
0.2	0.05	λ_1	0.0663	0.1019	0.1253	0.0834	0.0941	0.1107	0.0941	0.0954	0.1054
		β	0.0043	0.118	0.1488	0.005	0.0777	0.0999	0.0056	0.0548	0.0694
0.2	0.1	λ_1	0.0848	0.1088	0.133	0.0918	0.1	0.1162	0.0951	0.0962	0.1061
		β	0.0044	0.1149	0.1458	0.0046	0.0787	0.1	0.0056	0.0545	0.0689
0.2	0.15	λ_1	0.091	0.1118	0.1351	0.0926	0.1008	0.1166	0.0952	0.0961	0.106
		β	0.0036	0.116	0.1472	0.0044	0.078	0.0995	0.0051	0.0546	0.0692
0.2	0.2	λ_1	0.0925	0.1131	0.1363	0.0929	0.1005	0.1165	0.0954	0.0964	0.1063

Tables 3.2 and 3.3 present the simulation results in case of omitting network 2 or network 1 from the estimation, respectively. Both cases exhibit similar patterns in terms of the bias of the parameter estimates. Peer effect parameters are always biased positively. The intuition behind this result is that the positive peer effects of the omitted network are picked up by the remaining,

DOI: 10.14754/CEU.2024.08

Table 3.3: Monte Carlo simulation results of the biased regression including only Network 2

Test case			**25 nodes**			**50 nodes**			**100 nodes**		
p network 1	p network 2	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	0.2533	0.2588	0.2888	0.1554	0.1634	0.1902	0.0568	0.077	0.0959
0.1	0.05	λ_2	0.0924	0.1142	0.1368	0.1846	0.1848	0.1992	0.2557	0.2557	0.2611
		β	0.1548	0.1819	0.2219	0.056	0.1012	0.1249	0.0265	0.062	0.0775
0.1	0.1	λ_2	0.1677	0.1761	0.2028	0.2538	0.2539	0.2655	0.2795	0.2795	0.284
		β	0.0861	0.1533	0.1929	0.0342	0.0907	0.1165	0.0199	0.0593	0.0752
0.1	0.15	λ_2	0.2157	0.2197	0.2465	0.2703	0.2703	0.2813	0.2848	0.2848	0.289
		β	0.0704	0.1475	0.1847	0.0276	0.0897	0.1155	0.0183	0.0584	0.0741
0.1	0.2	λ_2	0.2282	0.2304	0.2566	0.2755	0.2756	0.2867	0.2863	0.2863	0.2905
		β	0.2678	0.2716	0.2975	0.1512	0.1591	0.1848	0.0536	0.075	0.0936
0.2	0.05	λ_2	0.1047	0.1195	0.1391	0.1882	0.1882	0.202	0.2576	0.2576	0.2629
		β	0.1543	0.1812	0.2182	0.0476	0.0958	0.1189	0.0229	0.0612	0.0769
0.2	0.1	λ_2	0.1874	0.1915	0.2162	0.2601	0.2602	0.2714	0.2816	0.2816	0.2862
		β	0.0792	0.1442	0.1794	0.0255	0.0872	0.1116	0.016	0.0595	0.075
0.2	0.15	λ_2	0.2387	0.2401	0.262	0.2768	0.2768	0.2872	0.2872	0.2872	0.2914
		β	0.056	0.1362	0.1686	0.019	0.0867	0.1101	0.0145	0.0577	0.0737
0.2	0.2	λ_2	0.2559	0.2565	0.2777	0.2819	0.2821	0.2925	0.2886	0.2886	0.2928

included network. The pattern in the bias across different test cases also supports this finding. We can observe that holding a network density (p_1 or p_2) fixed, increasing the other network's density - and thus the overlap between the two networks - increases the bias of the peer effect estimates.

Another observable pattern across all test cases is that increasing the sample size results in a more precise identification of the exogenous covariate's regression coefficient. This, in turn, results in a higher bias for the estimated peer effect parameter as the effects of the omitted network are picked up by this parameter instead.

3.3.2 Simulation using Barabasi-Albert graphs

This section explores how the estimator performs using Barabasi-Albert random graphs (Albert and Barabási, 2002). These graphs are generated using a preferential attachment mechanism, where nodes sequentially enter the network and form exactly m new connections randomly. The probability of forming a link with any pre-existing node i that is already in the network is proportional to the degree of the given node. That is, the new node is more likely to connect with already well-connected agents.

In order to control for the overlap between the two underlying networks, we apply the

DOI: 10.14754/CEU.2024.08

following logic in this experiment:

- We start generating a single network using the Barabasi-Albert method.
- Once the number of nodes in the network reaches a pre-defined fraction of the total number of nodes to be added, we create a copy of this network.
- The two networks are then completed separately.

The method outlined above ensures that the two networks overlap, and provides a way to control the extent of this overlap by changing the cutoff point where the networks are separated. We run our simulations using the following parameter space:

- m: the number of new connections each new node forms. We evaluate two cases, $m \in \{2, 4\}$.
- Cutoff percentage: we analyze our method using four distinct cutoff points: 20%, 30%, 40% and 50%.
- Number of nodes: similarly to our previous experiments, we use 25, 50 and 100 nodes.

Table 3.4 presents the estimation results with both networks simultaneously included in the regression. The patterns we can observe are similar to those of the Erdos-Renyi case. Our estimates have a low level of overall bias. The magnitude of the bias shows a negative association with the number of nodes. Having higher number of edges per node entering is positively correlated with the magnitude of the bias. This is intuitive, as in more dense networks the expected overlap between the two networks is higher. The same intuition applies when it comes to the cutoff percentage, which also shows a positive association with the magnitude of the bias.

Our simulation results omitting either the first or the second network from the estimation are presented in Table 3.5 and Table 3.6. These results are qualitatively identical to those of the experiment using Erdos-Renyi graphs. All estimates showcase a significant bias. The magnitude

DOI: 10.14754/CEU.2024.08

Table 3.4: Monte Carlo simulation results of the regression including both networks simultaneously using Barabasi-Albert graphs

Test case			**25 nodes**			**50 nodes**			**100 nodes**		
m	cutoff	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
2	20%	β	-0.0122	0.1202	0.1522	-0.0077	0.0797	0.1008	0.0003	0.0538	0.0689
		λ_1	0.0037	0.1612	0.2133	0.0026	0.1067	0.1335	-0.0012	0.0703	0.0881
		λ_2	-0.0011	0.1579	0.2051	-0.0016	0.1024	0.1282	-0.0002	0.0699	0.0877
2	30%	β	-0.0137	0.1175	0.149	-0.0089	0.0787	0.1005	0.0005	0.0548	0.0696
		λ_1	-0.0018	0.1734	0.2247	0.001	0.1096	0.1398	-0.0041	0.069	0.0866
		λ_2	0.0055	0.1678	0.22	0.001	0.1079	0.1364	0.0027	0.0719	0.0896
2	40%	β	-0.0124	0.121	0.153	-0.0076	0.08	0.1014	-0.0001	0.0549	0.07
		λ_1	0.0024	0.1814	0.2438	0.0044	0.1113	0.142	-0.0037	0.073	0.0922
		λ_2	0.0013	0.1794	0.2375	-0.0037	0.1123	0.1422	0.0029	0.0744	0.0937
2	50%	β	-0.0147	0.1168	0.1483	-0.0065	0.0806	0.1023	0.0003	0.0551	0.0695
		λ_1	-0.0048	0.1979	0.2472	0.0088	0.116	0.147	-0.0031	0.084	0.1057
		λ_2	0.0107	0.1916	0.2421	-0.0084	0.1119	0.1448	0.002	0.0858	0.107
4	20%	β	-0.0127	0.118	0.1462	-0.0078	0.0817	0.1038	-0.0005	0.0544	0.0702
		λ_1	0.0181	0.2488	0.3104	0.0124	0.1523	0.1922	-0.0016	0.1061	0.1336
		λ_2	-0.0141	0.2383	0.3001	-0.0113	0.1519	0.1929	0.0008	0.1076	0.1345
4	30%	β	-0.014	0.1183	0.1488	-0.0071	0.0803	0.1031	-0.0003	0.0548	0.0699
		λ_1	0.0105	0.2628	0.3309	0.0135	0.1611	0.2028	0.0074	0.1119	0.14
		λ_2	-0.0053	0.2577	0.3238	-0.0122	0.1579	0.2006	-0.0083	0.1106	0.1384
4	40%	β	-0.0117	0.1184	0.1502	-0.0079	0.0813	0.1044	0.0001	0.0551	0.0698
		λ_1	0.0043	0.2654	0.3401	0.0186	0.1697	0.2128	0.0019	0.1096	0.1396
		λ_2	-0.0004	0.2613	0.3383	-0.018	0.1708	0.2168	-0.0031	0.1097	0.1394
4	50%	β	-0.0137	0.119	0.1503	-0.0081	0.0809	0.1029	0.0002	0.054	0.0691
		λ_1	0.0156	0.2614	0.3289	0.0154	0.1675	0.2109	0.0065	0.1226	0.1517
		λ_2	-0.0097	0.2579	0.3281	-0.0136	0.1642	0.2097	-0.0078	0.1169	0.1455

Table 3.5: Monte Carlo simulation results of the biased regression including only Network 1 using Barabasi-Albert graphs

Test case			**25 nodes**			**50 nodes**			**100 nodes**		
m	cutoff	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
2	20%	β	0.0101	0.1165	0.1466	0.0129	0.0807	0.1032	0.0196	0.0577	0.0729
		λ_1	0.0866	0.1096	0.135	0.0853	0.0934	0.1107	0.0834	0.0854	0.096
2	30%	β	0.0093	0.1151	0.1451	0.013	0.0806	0.1028	0.0187	0.0566	0.0719
		λ_1	0.0877	0.1092	0.1344	0.0855	0.0936	0.1105	0.0841	0.086	0.0965
2	40%	β	0.0088	0.1165	0.146	0.0114	0.0795	0.1012	0.0173	0.0559	0.071
		λ_1	0.0879	0.1095	0.1351	0.0865	0.094	0.1108	0.0852	0.087	0.0972
2	50%	β	0.0075	0.1164	0.1452	0.0097	0.0807	0.1019	0.0155	0.0558	0.0705
		λ_1	0.0893	0.1108	0.1361	0.088	0.0953	0.1123	0.0869	0.0882	0.0986
4	20%	β	0.0008	0.116	0.145	0.0036	0.0792	0.1015	0.0101	0.0556	0.0704
		λ_1	0.0957	0.115	0.1386	0.0933	0.101	0.1177	0.0912	0.0924	0.1026
4	30%	β	0.0001	0.1153	0.1444	0.0032	0.0787	0.1005	0.0092	0.0559	0.0706
		λ_1	0.096	0.1151	0.1389	0.0936	0.1009	0.1175	0.0918	0.093	0.1032
4	40%	β	0.0001	0.1151	0.1445	0.0029	0.0794	0.101	0.0086	0.0555	0.0702
		λ_1	0.096	0.1154	0.1388	0.094	0.1013	0.118	0.0921	0.0932	0.1034
4	50%	β	0.0007	0.1148	0.1445	0.0023	0.0789	0.1007	0.0085	0.0555	0.0701
		λ_1	0.0957	0.115	0.1386	0.0942	0.1015	0.1181	0.0925	0.0937	0.1037

of the bias is positively correlated with the peer effect of the omitted network, the cutoff point and the density of the networks. Including more observations marginally decreases the bias, however, even using 100 nodes it remains substantial.

DOI: 10.14754/CEU.2024.08

Table 3.6: Monte Carlo simulation results of the biased regression including only Network 2 using Barabasi-Albert graphs

Test case			25 nodes			50 nodes			100 nodes		
m	cutoff	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	0.0507	0.1338	0.167	0.0602	0.0993	0.1237	0.0717	0.0879	0.1081
2	20%	λ_2	0.2567	0.2578	0.2795	0.2506	0.2506	0.262	0.2435	0.2435	0.2499
		β	0.0473	0.1314	0.1656	0.0547	0.0952	0.1198	0.0678	0.0847	0.1041
2	30%	λ_2	0.2583	0.2609	0.2813	0.2543	0.2543	0.2657	0.2467	0.2467	0.2527
		β	0.0455	0.1316	0.1647	0.0519	0.0966	0.1218	0.0602	0.0828	0.1012
2	40%	λ_2	0.2606	0.2634	0.2843	0.2561	0.2561	0.268	0.2528	0.2528	0.2588
		β	0.0377	0.1293	0.1622	0.0499	0.0922	0.1174	0.0551	0.0768	0.0947
2	50%	λ_2	0.2666	0.2679	0.2882	0.2582	0.2583	0.2693	0.2563	0.2563	0.2617
		β	0.023	0.1197	0.1497	0.0309	0.0896	0.1125	0.0354	0.0668	0.0845
4	20%	λ_2	0.278	0.2789	0.2966	0.2726	0.2727	0.2836	0.2725	0.2725	0.2776
		β	0.0221	0.1198	0.1503	0.0285	0.0869	0.1122	0.036	0.0654	0.0823
4	30%	λ_2	0.28	0.2813	0.2988	0.2746	0.2746	0.2851	0.2723	0.2723	0.2771
		β	0.0214	0.1196	0.1521	0.027	0.0874	0.1125	0.0339	0.0656	0.0818
4	40%	λ_2	0.2801	0.2814	0.2998	0.2753	0.2754	0.2862	0.2743	0.2743	0.2792
		β	0.0167	0.1201	0.1506	0.0235	0.0852	0.1082	0.0296	0.0626	0.0783
4	50%	λ_2	0.283	0.2838	0.302	0.2789	0.2792	0.2892	0.2768	0.2768	0.2813

3.3.3 Simulation using networks with homophily

Table 3.7: Monte Carlo simulation results of the regression including both networks simultaneously using networks with homophily

Test case			25 nodes			50 nodes			100 nodes		
m	cutoff	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	-0.0117	0.1528	0.1958	-0.0118	0.1075	0.1351	-0.0082	0.0775	0.0959
2	20%	λ_1	0.0126	0.1682	0.2158	0.0152	0.1093	0.1399	0.0003	0.0759	0.093
		λ_2	-0.0092	0.1634	0.2097	-0.0112	0.1011	0.128	0.0039	0.0756	0.094
		β	-0.0228	0.1493	0.1912	-0.0105	0.1037	0.1311	-0.0065	0.0762	0.0945
2	30%	λ_1	0.0081	0.1756	0.2274	0.0065	0.1059	0.1357	0.0082	0.078	0.0958
		λ_2	0.0026	0.1722	0.2237	-0.0025	0.1008	0.128	-0.0054	0.0732	0.0922
		β	-0.0141	0.1506	0.1942	-0.0098	0.1064	0.1332	-0.0061	0.0756	0.0941
2	40%	λ_1	0.016	0.1786	0.2338	0.0092	0.1135	0.1443	0.0089	0.0807	0.1016
		λ_2	-0.0125	0.1732	0.2244	-0.0058	0.1081	0.1369	-0.0066	0.0787	0.0974
		β	-0.014	0.1483	0.1916	-0.0088	0.1048	0.1307	-0.0089	0.0747	0.094
2	50%	λ_1	-0.0001	0.1807	0.2515	0.0103	0.124	0.1557	0.0019	0.0814	0.1033
		λ_2	0.0051	0.1786	0.2423	-0.0074	0.1154	0.1495	0.0024	0.0798	0.1014
		β	-0.013	0.1531	0.1924	-0.0053	0.1039	0.1309	-0.0064	0.0777	0.0949
4	20%	λ_1	0.0059	0.2494	0.3221	0.0076	0.1503	0.1923	-0.002	0.1031	0.1303
		λ_2	-0.0025	0.2496	0.3262	-0.0076	0.1463	0.1868	0.0046	0.1032	0.1315
		β	-0.0191	0.149	0.1886	-0.0062	0.1028	0.1288	-0.0048	0.0772	0.0955
4	30%	λ_1	0.005	0.2525	0.3327	0.0096	0.158	0.1993	0.0112	0.099	0.1242
		λ_2	0.0027	0.2491	0.33	-0.0092	0.1574	0.1986	-0.0096	0.0991	0.1248
		β	-0.0173	0.1484	0.1876	-0.0052	0.1031	0.1289	-0.0025	0.0768	0.0946
4	40%	λ_1	-0.0062	0.2383	0.3132	0.0076	0.1617	0.2044	0.0059	0.1166	0.1462
		λ_2	0.0124	0.2403	0.3153	-0.0083	0.1589	0.1997	-0.006	0.1169	0.1453
		β	-0.019	0.1463	0.1912	-0.0035	0.104	0.1307	-0.0041	0.0784	0.0965
4	50%	λ_1	-0.0121	0.2715	0.3421	0.0092	0.1549	0.1989	-0.0036	0.1142	0.1433
		λ_2	0.0196	0.2715	0.3476	-0.0105	0.1542	0.1981	0.0047	0.115	0.1438

The last set of simulations considers a case where networks show homophily, that is, the network connections of agents are dependent on their characteristics. This phenomenon is

DOI: 10.14754/CEU.2024.08

present in many real-life networks: agents are more likely to connect with other, similar agents. In order to simulate such a scenario, we have modified the mechanism of the Barabasi-Albert model. Our logic of generating the networks in this experiment is the following: nodes enter the network one-by-one and form exactly m new connections randomly, just like in the Barabasi-Albert model. However, the probability of forming a link with any pre-existing node i that is already in the network is proportional to their absolute distance in the covariate associated with each node. This means that the new node will be more likely to connect with another agent that has similar characteristics.

All other components of this experiment imitate the previous one using Barabasi-Albert graphs. We evaluate cases with $m \in \{2, 4\}$ connections, 20%, 30%, 40% and 50% cutoffs, and 25, 50 and 100 nodes.

The results of the simulations where both networks are included in the estimation are presented in Table 3.7. The findings of the previous exercises hold in this case as well. All estimates are characterized with a low relative bias, and the direction of the association between the magnitude of the bias and the hyperparameters are unchanged.

Estimates that rely on a single network are presented in Table 3.8 and Table 3.9. The qualitative results presented for the two other cases hold here as well. Peer effect estimates are characterized by significant positive bias in both cases and the bias is more severe if (i) the networks are more dense, (ii) the overlap between the networks is higher or (iii) the networks are smaller, and thus the number of observations is lower.

3.4 Empirical example

This section presents an illustrative example using data from Ferrali et al. (2020). The next section provides background on the data used for peer effect estimations. It is followed by discussing the results of estimating the peer effects using three strategies: (i) estimating peer effects from a collapsed network; (ii) estimating peer effects network-by-network and (iii)

DOI: 10.14754/CEU.2024.08

Table 3.8: Monte Carlo simulation results of the biased regression including only Network 1 using networks with homophily

Test case			25 nodes			50 nodes			100 nodes		
m	cutoff	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	0.0074	0.1164	0.1455	0.0126	0.0828	0.1041	0.0176	0.058	0.0739
2	20%	λ_1	0.09	0.1107	0.1344	0.0857	0.0949	0.1121	0.0849	0.0867	0.0973
		β	0.0088	0.1151	0.1448	0.0116	0.0822	0.1034	0.0167	0.058	0.0737
2	30%	λ_1	0.0895	0.1101	0.1332	0.0864	0.0949	0.1123	0.0856	0.0873	0.0979
		β	0.0073	0.1151	0.1438	0.0109	0.0813	0.1032	0.0162	0.0575	0.0735
2	40%	λ_1	0.0908	0.1102	0.1336	0.0871	0.0959	0.1132	0.0861	0.0877	0.0984
		β	0.0063	0.1145	0.144	0.0099	0.0814	0.103	0.0153	0.0569	0.0724
2	50%	λ_1	0.0916	0.1114	0.1349	0.0879	0.0962	0.1135	0.087	0.0885	0.099
		β	0.0022	0.1134	0.1424	0.0039	0.0796	0.101	0.0103	0.0568	0.0713
4	20%	λ_1	0.0949	0.1136	0.1353	0.0929	0.101	0.1169	0.0908	0.0922	0.1028
		β	0.0008	0.1134	0.1417	0.0032	0.0796	0.1006	0.01	0.0572	0.072
4	30%	λ_1	0.0957	0.1138	0.1355	0.0936	0.1009	0.1171	0.0912	0.0926	0.1033
		β	0.0016	0.1134	0.1432	0.0023	0.0796	0.1004	0.0097	0.0571	0.0715
4	40%	λ_1	0.0953	0.1137	0.1356	0.094	0.1016	0.1177	0.0914	0.0929	0.1036
		β	0.0003	0.1139	0.1424	0.0026	0.08	0.1008	0.0093	0.0574	0.0718
4	50%	λ_1	0.0962	0.1144	0.1364	0.0941	0.1014	0.1175	0.0919	0.0932	0.1039

Table 3.9: Monte Carlo simulation results of the biased regression including only Network 2 using networks with homophily

Test case			25 nodes			50 nodes			100 nodes		
m	cutoff	Parameter	BIAS	MAE	RMSE	BIAS	MAE	RMSE	BIAS	MAE	RMSE
		β	0.0536	0.1348	0.1718	0.0635	0.1033	0.131	0.0704	0.0866	0.1047
2	20%	λ_2	0.2557	0.2572	0.2789	0.2487	0.2487	0.261	0.2461	0.2461	0.2516
		β	0.0456	0.1401	0.1747	0.061	0.1048	0.1313	0.0649	0.0831	0.1016
2	30%	λ_2	0.2627	0.2636	0.2856	0.2503	0.2505	0.263	0.2502	0.2502	0.2558
		β	0.0463	0.1376	0.1742	0.0541	0.0991	0.1257	0.0617	0.0807	0.0999
2	40%	λ_2	0.2618	0.2626	0.285	0.2557	0.2558	0.2674	0.252	0.252	0.2578
		β	0.0382	0.1321	0.1666	0.0482	0.0955	0.1216	0.0535	0.0754	0.094
2	50%	λ_2	0.2681	0.2688	0.2889	0.261	0.2613	0.2721	0.2587	0.2587	0.2638
		β	0.0189	0.1162	0.1476	0.0286	0.0916	0.1145	0.0352	0.0662	0.0833
4	20%	λ_2	0.2826	0.2829	0.2999	0.2751	0.2752	0.2861	0.2732	0.2732	0.278
		β	0.0211	0.1198	0.1511	0.0266	0.0874	0.1112	0.0346	0.0664	0.0827
4	30%	λ_2	0.2809	0.2812	0.2996	0.2769	0.2769	0.2869	0.2733	0.2733	0.2781
		β	0.0173	0.1189	0.1506	0.0289	0.0883	0.1126	0.0307	0.0647	0.0813
4	40%	λ_2	0.2837	0.2838	0.3023	0.2757	0.2759	0.286	0.2763	0.2763	0.281
		β	0.0184	0.1178	0.1486	0.0211	0.0846	0.1075	0.0288	0.0631	0.0789
4	50%	λ_2	0.2831	0.2835	0.301	0.2811	0.2811	0.2906	0.2776	0.2776	0.2822

estimating peer effects simultaneously. We then discuss how the estimation results relate to the Monte Carlo simulation results presented above and point out the main takeaways from applying different strategies.

3.4.1 Data

We use survey data collected by Ferrali et al. (2020) in Uganda 2 years after the introduction of a new communication platform (U-Bridge) connecting district officials and citizens. The

DOI: 10.14754/CEU.2024.08

technology was implemented in a northwestern region of Uganda, Arua. The platform itself consisted of a short-code number to which any citizen could send messages to report disruptions in government-provided services free of charge, anonymously. On the other end of the communication, officials were given tablets to access and respond to the reports from citizens.

Ferrali et al. (2020) conducted in-person surveys in April and May 2016. They attempted to collect information from every available adult in 16 villages. The surveys contained questions regarding demographics, social ties, and U-bridge knowledge and usage. They obtained information from 3,184 individuals, covering about 82%

They collected information about multiple social ties across individuals and constructed five networks between the same agents. These are the following:

- *Friendship network:* There is an edge pointing from node i to node j if i considers j a friend.
- *Family network:* There is an edge pointing from node i to node j if i names j as a family member.
- *Lender network:* There is an edge pointing from node i to node j if i named j as someone who would lend them money.
- *Problem solver network:* There is an edge pointing from node i to node j if i would go to j to solve a problem.
- *Speaking network:* There is an edge pointing from node i to node j if i spoke to j about the technology.

Additionally, they generated and used a network pooling all the edges from all the networks listed above. The networks collected by Ferrali et al. (2020) contain overlapping edges - in line with the model setup - meaning that individual i might consider individual j a friend, family member, lender, solver or could have spoken to them about the technology all at the same time. A measure of pairwise overlap is presented in Table 3.10. Table 3.10 shows that up to 33.6% of

DOI: 10.14754/CEU.2024.08

the edges in a network are present in another network as well (excluding the pooled network, where it can be by construction higher for the denser networks) which indicates significant overlap between the networks. Due to these overlaps, this dataset is optimal to analyze the effect of applying different estimation strategies and to evaluate the proposed estimator.

Table 3.10: Similarities of networks. Numbers show what fraction of the edges in the row network are present in the column network. The last row contains the number of edges in the given network.

	all	**family**	**friend**	**lender**	**solver**	**speak**
all	1.0	0.481	0.277	0.211	0.228	0.022
family	1.0	1.0	0.051	0.147	0.088	0.014
friend	1.0	0.088	1.0	0.182	0.072	0.016
lender	1.0	0.336	0.240	1.0	0.122	0.020
solver	1.0	0.187	0.087	0.113	1.0	0.015
speak	1.0	0.316	0.199	0.193	0.154	1.0
|E|	30,329	14,613	8,403	6,401	6,927	686

We estimate three outcomes:

1. *Heard*: Heard is an indicator that gets the value of 1 if the respondent has heard about the U-Bridge service.

2. *Adopt*: Adopt is a self-reported, binary variable that equals 1 if the respondent has used the platform at least once in the past 12 months.

3. *Sent*: Shows the level of usage measured by the self-reported number of messages sent by the individual in the past 6 months.

We estimate each model with and without controls as well. Controls are gender, age, income, education, and village indicators. Our evaluation strategy is analogous to the simulations presented in Section 3.3. That is, (i) we estimate peer effects using a network-by-network fashion, including only a single network in the estimation at a time and (ii) we estimate the peer effects for each network simultaneously using the proposed estimator. Additionally, we evaluate a strategy where we use the pooled network of social ties to estimate a single peer effect. The next section presents and discusses the results of each estimation strategy.

DOI: 10.14754/CEU.2024.08

3.4.2 Estimated peer effects

The estimated coefficients using the collapsed network and the networks one by one are presented in Table 3.11. Each coefficient estimate is underset with its corresponding standard error. The results show that every estimated peer effect parameter is significant at the one percent level for all outcome variables. Ferrali et al. (2020) also estimate peer effects in the collapsed network using a different approach. The peer effect they estimate is of the same magnitude as the one presented in Table 3.11. We can observe that the estimated peer effects are robust to the inclusion of controls.

Table 3.11: Estimated coefficients using separate networks for different outcome variables.

Parameter		All		Family		Friend		Lender		Solver		Speak	
Hear	λ	.022 .003	.021 .002	.056 .006	.044 .004	.077 .006	.077 .004	.069 .005	.056 .004	.027 .002	.017 .002	.198 .006	.156 .007
	ρ	.015 .003	.008 .003	−.002 .003	−.020 .006	−.035 .007	−.062 .007	−.019 .008	−.035 .008	−.027 .003	.004 .007	−.094 .013	−.073 .015
Adopt	λ	.056 .003	.049 .004	.083 .006	.089 .004	.112 .006	.120 .006	.104 .005	.111 .005	.051 .004	.060 .004	.230 .004	.225 .004
	ρ	−.004 .001	.007 .003	−.060 .005	−.072 .005	−.075 .006	−.091 .006	−.086 .006	−.098 .006	−.043 .003	−.040 .005	−.208 .004	−.216 .005
Sent	λ	.063 .007	.052 .005	.101 .007	.098 .003	.117 .013	.146 .005	.113 .005	.138 .003	.065 .007	.084 .005	.218 .005	.206 .005
	ρ	−.011 .001	−.011 .005	−.110 .003	−.097 .004	−.025 .005	−.339 .003	−.297 .002	−.239 .004	−.014 .002	−.121 .002	−.385 .003	−.103 .003
	Controls		✓		✓		✓		✓		✓		✓
	N	3019	3019	3019	3019	3019	3019	3019	3019	3019	3019	3019	3019

In the case of the simultaneous estimation, results are presented in Table 3.12. Compared to the network-by-network strategy, most estimated peer effect coefficients drop significantly. While the majority of them remain significant for all outcome variables, some of them even become insignificant.

This significant decrease in the estimated parameters is consistent with both our model presented in 3.2.2 and the Monte Carlo simulations. The intuition is that the networks we consider are overlapping, as shown in Table 3.10, and therefore estimating peer effects in a network-by-network fashion results in omitted network bias. Each peer effect estimated by the network-by-network strategy picks up partial effects of the correlated networks. This can lead to stronger, significant peer effect estimates in networks that, in fact, have less of a role in the

DOI: 10.14754/CEU.2024.08

Table 3.12: Estimated coefficients with simultaneous estimation for different outcome variables

Parameter		Hear		Adopt		Sent	
λ	Family	.002 (.006)	.012 (.004)	.011 (.005)	.010 (.004)	−.005 (.004)	.022 (.003)
	Friend	.038 (.007)	.039 (.005)	.008 (.006)	.018 (.005)	−.027 (.006)	.045 (.004)
	Lender	.014 (.006)	.010 (.005)	−.001 (.006)	.008 (.005)	.018 (.005)	.028 (.004)
	Solver	−.004 (.003)	−.005 (.002)	−.003 (.004)	−.002 (.003)	.023 (.003)	.017 (.003)
	Speak	.138 (.019)	.112 (.009)	.229 (.007)	.215 (.005)	.197 (.005)	.147 (.005)
ρ	Family	.037 (.007)	.005 (.006)	−.008 (.007)	−.011 (.005)	.037 (.006)	−.006 (.004)
	Friend	−.006 (.009)	−.030 (.008)	.006 (.008)	−.010 (.006)	.060 (.005)	−.030 (.005)
	Lender	−.006 (.009)	−.007 (.008)	.009 (.008)	−.002 (.006)	.024 (.005)	−.071 (.004)
	Solver	−.013 (.005)	.007 (.008)	.002 (.005)	.002 (.004)	−.057 (.003)	−.071 (.002)
	Speak	−.062 (.027)	−0.65 (.016)	−.212 (.007)	−.204 (.006)	−.334 (.003)	−.080 (.003)
Controls			✓		✓		✓
N		3019	3019	3019	3019	3019	3019

spillover. Taking a closer look at which parameters remained strongly significant and high-magnitude, we can observe that the only such parameter is the one belonging to the speaking network. This intuitively means that both knowledge about the technology and uptake are mostly influenced by talking about it, which is a fairly straightforward finding. At the same time, it is important to notice that a researcher estimating peer effects using the individual networks would reach a different conclusion, namely, that all kinds of social ties have a strong influence on both the spread of information and the uptake of the technology.

3.5 Conclusion

This paper proposed a tractable theoretical model with parameters that can be estimated using conventional tools from spatial econometrics to model and estimate peer effects in multiplex networks. This contribution can be particularly important as the failure to account for multiple

DOI: 10.14754/CEU.2024.08

correlated networks results in biased peer effect estimates. Our findings demonstrated the importance of including multiple networks in the estimation process.

The proposed model, despite its limitations, provides a valuable tool in the analysis of heterogeneous peer effects across different types of relationships. Given that individuals often interact with each other on different platforms or in diverse social settings, these complexities should be reflected in research models.

The Monte Carlo experiments that we ran provided compelling evidence of the potential biases that could result from using traditional, single-network models when estimating peer effects in a multiplex network setting. The results highlighted that such biases not only perturb the true relations among variables but may also overestimate the impacts of certain peer effects due to omitted networks.

Our empirical analysis utilizing data from Uganda provided an illustrative example of how peer effects could be misinterpreted when multiple types of relationships are not included in the estimation step. Using network-by-network estimation, we found that all types of social ties appeared to have a strong influence on both the spread of information and the uptake of the technology. However, when we controlled for multiple networks simultaneously, there was a significant decline in the peer effect coefficients, and some even became insignificant.

In conclusion, our work highlights the importance of incorporating multiplex networks in estimating peer effects. Revisiting the way we quantify and understand peer effects can provide researchers, policymakers, and social planners with insights needed to more accurately interpret social phenomena and design effective interventions and strategies.

DOI: 10.14754/CEU.2024.08

References

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. "Tensorflow: A System for Large-Scale Machine Learning." 265–283.

Acemoglu, Daron, Vasco M. Carvalho, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. 2012. "The Network Origins of Aggregate Fluctuations." *Econometrica*, 80(5): 1977–2016.

Adamic, Lada A, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. 2001. "Search in power-law networks." *Physical review E*, 64(4): 046135.

Ahmed, Nesreen K, Jennifer Neville, and Ramana Kompella. 2013. "Network sampling: From static to streaming graphs." *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2): 1–56.

Ahmed, Nesreen K, Ryan A Rossi, John Boaz Lee, Theodore L Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2019. "role2vec: Role-based network embeddings."

Albert, Réka, and Albert-László Barabási. 2002. "Statistical mechanics of complex networks." *Rev. Mod. Phys.*, 74: 47–97.

Albizuri, Josune, Jesús Aurrecoechea, et al. 2006. "Configuration Values: Extensions of the Coalitional Owen Value." *Games and Economic Behavior*, 1–17.

Ammermueller, Andreas, and Jörn-Steffen Pischke. 2009. "Peer Effects in European Primary Schools: Evidence from the Progress in International Reading Literacy Study." *Journal of Labor Economics*, 27(3): 315–348.

Ancona, Marco, Cengiz Oztireli, and Markus Gross. 2019. "Explaining Deep Neural Networks with a Polynomial Time Algorithm for Shapley Value Approximation." 272–281.

Anderson, Sean, Eric Ward, Lewis Barnett, and Philippina English. 2018. "sdmTMB: Spatial and spatiotemporal GLMMs with TMB." `https://github.com/pbs-assess/sdmTMB`.

Badinger, Harald, and Peter Egger. 2011. "Estimation of higher-order spatial autoregressive cross-section models with heteroscedastic disturbances." *Papers in Regional Science*, 90(1): 213–235.

Bahdanau, Dzmitry, Kyung Hyun Cho, and Yoshua Bengio. 2015. "Neural Machine Translation by Jointly Learning to Align and Translate."

Bai, Lei, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. 2020. "Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting." *Advances in Neural Information Processing Systems*, 33.

Ballester, Coralio, Antoni Calvó-Armengol, and Yves Zenou. 2006. "Who's Who in Networks. Wanted: The Key Player." *Econometrica*, 74(5): 1403–1417.

Bandyopadhyay, Sambaran, Harsh Kara, Aswin Kannan, and M Narasimha Murty. 2018. "Fscnmf: Fusing structure and content via non-negative matrix factorization for embedding information networks." *arXiv preprint arXiv:1804.05313*.

Banzhaf III, John F. 1964. "Weighted Voting Doesn't Work: A Mathematical Analysis." *Rutgers L. Rev.*, 317.

Belkin, Mikhail, and Partha Niyogi. 2002. "Laplacian eigenmaps and spectral techniques for embedding and clustering." 585–591.

Béres, Ferenc, Domokos M. Kelen, Róbert Pálovics, and András A. Benczúr. 2019. "Node Embeddings in Dynamic Graphs." *Applied Network Science*, 4(64): 25.

Béres, Ferenc, Róbert Pálovics, Anna Oláh, and András A. Benczúr. 2018. "Temporal Walk Based Centrality Metric for Graph Streams." *Applied Network Science*, 3(32): 26.

Bojchevski, Aleksandar, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. "Scaling Graph Neural Networks with Approximate Pagerank." 2464–2473.

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. "JAX: Composable Transformations of Python+NumPy Programs."

Brandl, Georg. 2010. "Sphinx Documentation." *URL http://sphinx-doc. org/sphinx. pdf*.

Buitinck, Lars, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jacob VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. "API design for machine learning software: experiences from the scikit-learn project." *ArXiv*, abs/1309.0238.

Burgess, Mark A., and Archie C. Chapman. 2021. "Approximating the Shapley Value Using Stratified Empirical Bernstein Sampling." 73–81. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Cao, Shaosheng, Wei Lu, and Qiongkai Xu. 2015. "Grarep: Learning graph representations with global structural information." 891–900, ACM.

DOI: 10.14754/CEU.2024.08

Castro, Javier, Daniel Gómez, and Juan Tejada. 2009. "Polynomial Calculation of the Shapley Value Based on Sampling." *Computers & Operations Research*, 36(5): 1726–1730.

Castro, Javier, Daniel Gómez, et al. 2017. "Improving Polynomial Estimation of the Shapley Value by Stratified Random Sampling with Optimum Allocation." *Computers & Operations Research*, 82: 180–188.

Centola, Damon. 2010. "The Spread of Behavior in an Online Social Network Experiment." *Science*, 329(5996): 1194–1197.

Cen, Yukuo, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Peng Zhang, Guohao Dai, et al. 2021. "CogDL: An Extensive Toolkit for Deep Learning on Graphs." *arXiv preprint arXiv:2103.00959*.

Chalkiadakis, Georgios, Edith Elkind, and Michael Wooldridge. 2011. "Computational Aspects of Cooperative Game Theory." *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(6): 1–168.

Chen, Hong, and Hisashi Koga. 2019. "GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features." 3–14, Springer.

Chen, Jianbo, Le Song, Martin Wainwright, and Michael Jordan. 2018*a*. "L-Shapley and C-Shapley: Efficient Model Interpretation for Structured Data."

Chen, Jinyin, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. 2018*b*. "GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction." *arXiv preprint arXiv:1812.04206*.

Chen, Tianqi, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems." *arXiv preprint arXiv:1512.01274*.

Cho, Kyunghyun, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. "Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation." 1724–1734. Association for Computational Linguistics.

Cohen, Shay, Gideon Dror, and Eytan Ruppin. 2007. "Feature Selection via Coalitional Game Theory." *Neural Computation*, 19(7): 1939–1961.

Coleman, James S. 1988. "Social Capital in the Creation of Human Capital." *American Journal of Sociology*, 94: S95–S120.

Cook, R Dennis. 1977. "Detection of influential observation in linear regression." *Technometrics*.

Covert, Ian, and Su-In Lee. 2021. "Improving KernelSHAP: Practical Shapley Value Estimation Using Linear Regression." 3457–3465.

DOI: 10.14754/CEU.2024.08

Dasgupta, Anirban, Petros Drineas, , et al. 2009. "Sampling algorithms and coresets for \ell_p regression." *SIAM Journal on Computing*, 2060–2078.

Data61, CSIRO's. 2018. "StellarGraph Machine Learning Library." *https://github.com/stellargraph/stellargraph*.

Davis, Gerald, Mina Yoo, and Wayne Baker. 2003. "The Small World of the American Corporate Elite, 1982-2001." *Strategic Organization*, 1.

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. 2016. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering." 3844–3852.

Defferrard, Michaël, Lionel Martin, Rodrigo Pena, and Nathanaël Perraudin. 2017. "PyGSP: Graph Signal Processing in Python."

de Lara, Nathan, and Pineau Edouard. 2018. "A simple baseline algorithm for graph classification."

Deutch, Daniel, Nave Frost, Amir Gilad, and Oren Sheffer. 2021. "Explanations for Data Repair Through Shapley Values." *CIKM '21*, 362–371. New York, NY, USA:Association for Computing Machinery.

Doerr, Christian, and Norbert Blenn. 2013. "Metric convergence in social network sampling." 45–50.

Donnat, Claire, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. "Learning structural node embeddings via diffusion wavelets." 1320–1329, ACM.

Duval, Alexandre, and Fragkiskos D. Malliaros. 2021. "GraphSVX: Shapley Value Explanations for Graph Neural Networks." 302–318.

Easley, David, Jon Kleinberg, et al. 2010. *Networks, crowds, and markets.* Vol. 8, Cambridge university press Cambridge.

Epasto, Alessandro, Silvio Lattanzi, and Renato Paes Leme. 2017. "Ego-Splitting Framework: From Non-Overlapping to Overlapping Clusters." *KDD '17*, 145–154.

Fan, Zhenan, Huang Fang, Zirui Zhou, et al. 2021. "Improving Fairness for Data Valuation in Federated Learning." *arXivv:2109.09046*.

Fatima, Shaheen S, Michael Wooldridge, and Nicholas R Jennings. 2008. "A Linear Approximation Method for the Shapley Value." *Artificial Intelligence*, 172(14): 1673–1699.

Ferrali, Romain, Guy Grossman, Melina R. Platas, and Jonathan Rodden. 2020. "It Takes a Village: Peer Effects and Externalities in Technology Adoption." *American Journal of Political Science*, 64(3): 536–553.

Fey, Matthias, and Jan E. Lenssen. 2019. "Fast Graph Representation Learning with PyTorch Geometric."

Fortin, Bernard, and Myra Yazbeck. 2015. "Peer effects, fast food consumption and adolescent weight gain." *Journal of Health Economics*, 42: 125–138.

Frostig, Roy, Matthew James Johnson, and Chris Leary. 2018. "Compiling Machine Learning Programs via High-Level Tracing." *Systems for Machine Learning*.

Frye, Christopher, Colin Rowat, and Ilya Feige. 2020. "Asymmetric Shapley Values: Incorporating Causal Knowledge Into Model-Agnostic Explainability." *Advances in Neural Information Processing Systems*, 33.

Frye, Christopher, Damien de Mijolla, Tom Begley, et al. 2020. "Shapley Explainability on the Data Manifold."

Fryer, Daniel, Inga Strümke, and Hien Nguyen. 2021. "Shapley Values for Feature Selection: the Good, the Bad, and the Axioms." *arXiv:2102.10936*.

Galeotti, Andrea, Benjamin Golub, and Sanjeev Goyal. 2020. "Targeting Interventions in Networks." *Econometrica*, 88(6): 2445–2471.

Gao, Feng, Guy Wolf, and Matthew Hirn. 2019. "Geometric Scattering for Graph Data Analysis." Vol. 97, 2122–2131.

Ghorbani, Amirata, and James Zou. 2019. "Data Shapley: Equitable Valuation of Data for Machine Learning." 2242–2251.

Ghorbani, Amirata, and James Zou. 2020. "Neuron Shapley: Discovering the Responsible Neurons." 5922–5932.

Ghorbani, Amirata, Michael Kim, and James Zou. 2020. "A Distributional Framework for Data Valuation." 3535–3544.

Gilmer, Justin, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. "Neural Message Passing for Quantum Chemistry." 1263–1272, PMLR.

Gjoka, Minas, Carter T Butts, Maciej Kurant, and Athina Markopoulou. 2011. "Multigraph sampling of online social networks." *IEEE Journal on Selected Areas in Communications*, 29(9): 1893–1905.

Gjoka, Minas, Maciej Kurant, Carter T Butts, and Athina Markopoulou. 2010. "Walking in facebook: A case study of unbiased sampling of osns." 1–9, Ieee.

Godwin*, Jonathan, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. 2020. "Jraph: A Library for Graph Neural Networks in Jax."

Gonzalez, Joseph E, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. "Powergraph: Distributed graph-parallel computation on natural graphs." 17–30.

Goodman, Leo A. 1961. "Snowball sampling." *The annals of mathematical statistics*, 148–170.

DOI: 10.14754/CEU.2024.08

Goyal, Palash, and Emilio Ferrara. 2018. "GEM: A Python Package for Graph Embedding Methods." *Journal of Open Source Software*, 3(29): 876.

Goyal, Palash, Sujit Rokka Chhetri, Ninareh Mehrabi, Emilio Ferrara, and Arquimedes Canedo. 2018. "DynamicGEM: A Library for Dynamic Graph Embedding Methods." *arXiv preprint arXiv:1811.10734*.

Grattarola, Daniele, and Cesare Alippi. 2020. "Graph Neural Networks in TensorFlow and Keras with Spektral." *arXiv preprint arXiv:2006.12138*.

Grover, Aditya, and Jure Leskovec. 2016. "node2vec: Scalable feature learning for networks." 855–864.

Guha, Ritam, Ali Hussain Khan, et al. 2021. "CGA: A new feature selection model for visual human action recognition." *Neural Computing and Applications*, 33(10): 5267–5286.

Gulati, Ranjay, and Martin Gargiulo. 1999. "Where Do Interorganizational Networks Come From?" *American Journal of Sociology*, 104(5): 1439–1493.

Guo, Shengnan, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. "Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting." Vol. 33, 922–929.

Guyon, Isabelle, and André Elisseeff. 2003. "An Introduction to Variable and Feature Selection." *Journal of machine learning research*, 3(Mar): 1157–1182.

Gwozdz, Wencke, Alfonso Sousa-Poza, Lucia A. Reisch, Karin Bammann, Gabriele Eiben, Yiannis Kourides, Éva Kovács, Fabio Lauria, Kenn Konstabel, Alba M. Santaliestra-Pasias, Krishna Vyncke, and Iris Pigeot. 2015. "Peer effects on obesity in a sample of European children." *Economics and Human Biology*, 18: 139–152.

Hagberg, Aric, Pieter Swart, and Daniel S Chult. 2008. "Exploring network structure, dynamics, and function using NetworkX." Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Hamilton, William L, Rex Ying, and Jure Leskovec. 2017. "Representation learning on graphs: Methods and applications." *arXiv preprint arXiv:1709.05584*.

Hanson, Matthew. 2019. "The Open-Source Software Ecosystem for Leveraging Public Datasets in Spatio-Temporal Asset Catalogs (STAC)." Vol. 2019, IN23B–07.

Helpman, Elhanan, Marc Melitz, and Yona Rubinstein. 2008. "Estimating Trade Flows: Trading Partners and Trading Volumes." *The Quarterly Journal of Economics*, 123(2): 441–487.

Henderson, Keith, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. "Rolx: structural role extraction & mining in large graphs." 1231–1239.

Heskes, Tom, Evi Sijben, Ioan Gabriel Bucur, and Tom Claassen. 2020. "Causal Shapley Values: Exploiting Causal Knowledge to Explain Individual Predictions of Complex Models." *Advances in Neural Information Processing Systems*, 33.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural computation*, 9(8): 1735–1780.

Holme, Petter. 2015. "Modern Temporal Network Theory: A Colloquium." *The European Physical Journal B*, 88(9): 1–30.

Holme, Petter, and Jari Saramäki. 2012. "Temporal Networks." *Physics reports*, 519(3): 97–125.

Hsieh, Chih-Sheng, and Hans van Kippersluis. 2018. "Smoking initiation: Peers and personality." *Quantitative Economics*, 9(2): 825–863.

Hsieh, Chih-Sheng, and Lung Fei Lee. 2016. "A Social Interactions Model with Endogenous Friendship Formation and Selectivity." *Journal of Applied Econometrics*, 31(2): 301–319.

Hübler, Christian, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani. 2008. "Metropolis algorithms for representative subgraph sampling." 283–292, IEEE.

Hu, Jun, Shengsheng Qian, Quan Fang, Youze Wang, Quan Zhao, Huaiwen Zhang, and Changsheng Xu. 2021. "Efficient Graph Deep Learning in TensorFlow with TF Geometric." *arXiv preprint arXiv:2101.11552*.

Hu, Pili, and Wing Cheong Lau. 2013. "A survey and taxonomy of graph sampling." *arXiv preprint arXiv:1308.5865*.

Illés, Ferenc, and Péter Kerényi. 2019. "Estimation of the Shapley Value by Ergodic Sampling." *arXiv:1906.05224*.

Jackson, Matthew O., Brian W. Rogers, and Yves Zenou. 2017. "The Economic Consequences of Social-Network Structure." *Journal of Economic Literature*, 55(1): 49–95.

Jia, Ruoxi, David Dao, Boxin Wang, Hubis, et al. 2019. "Towards Efficient Data Valuation Based on the Shapley Value." 1167–1176.

Jundong Li, Liang Wu, Huan Liu. 2019. "Multi-Level Network Embedding with Boosted Low-Rank Matrix Approximation." 50–56, ACM.

Kang, U, Charalampos E Tsourakakis, and Christos Faloutsos. 2009. "Pegasus: A peta-scale graph mining system implementation and observations." 229–238, IEEE.

Kingma, Diederik, and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization."

Kipf, Thomas N., and Max Welling. 2017. "Semi-Supervised Classification with Graph Convolutional Networks."

DOI: 10.14754/CEU.2024.08

Koh, Pang Wei, and Percy Liang. 2017. "Understanding black-box predictions via influence functions." 1885–1894.

Krishnamurthy, Vaishnavi, Michalis Faloutsos, Marek Chrobak, Li Lao, J-H Cui, and Allon G Percus. 2005. "Reducing large internet topologies for faster simulations." 328–341, Springer.

Kuang, Da, Chris Ding, and Haesun Park. 2012. "Symmetric nonnegative matrix factorization for graph clustering." 106–117, SIAM.

Kumar, Elizabeth, Suresh Venkatasubramanian, Carlos Scheidegger, and Sorelle Friedler. 2020. "Problems with Shapley-Value-Based Explanations as Feature Importance Measures." 5491–5500.

Kwon, Yongchan, and James Zou. 2021. "Beta Shapley: a Unified and Noise-reduced Data Valuation Framework for Machine Learning." *arXiv:2110.14049*.

Kwon, Yongchan, Manuel A Rivas, and James Zou. 2021. "Efficient Computation and Analysis of Distributional Shapley Values." 793–801.

König, Michael D., Xiaodong Liu, and Yves Zenou. 2019. "R&D Networks: Theory, Empirics, and Policy Implications." *The Review of Economics and Statistics*, 101(3): 476–491.

Lazarsfeld, Paul Felix, and Robert K. Merton. 1954. "Friendship as a social process: a substantive and method-ological analysis." *Freedom and Control in Modern Society*, 18–66.

Lee, Chul-Ho, Xin Xu, and Do Young Eun. 2012. "Beyond random walk and metropolis-hastings samplers: why you should not backtrack for unbiased graph sampling." *ACM SIGMETRICS Performance evaluation review*, 40(1): 319–330.

Lee, Lung-fei, and Xiaodong Liu. 2010. "Efficient GMM Estimation of High Order Spatial Autoregressive Models with Autoregressive Disturbances." *Econometric Theory*, 26(1): 187–230.

Lee, Lung-fei, Xiaodong Liu, and Xu Lin. 2010. "Specification and estimation of social interaction models with network structures." *The Econometrics Journal*, 13(2): 145–176.

Lee, Lung-fei. 2003. "Best Spatial Two-Stage Least Squares Estimators for a Spatial Autoregressive Model with Autoregressive Disturbances." *Econometric Reviews*, 22(4): 307–335.

Leskovec, Jure, and Andrej Krevl. 2014. "SNAP Datasets: Stanford Large Network Dataset Collection." `http://snap.stanford.edu/data`.

Leskovec, Jure, and Christos Faloutsos. 2006. "Sampling from large graphs." 631–636.

Leskovec, Jure, Jon Kleinberg, and Christos Faloutsos. 2005. "Graphs over time: densification laws, shrinking diameters and possible explanations." 177–187.

Li, Jhao-Yin, and Mi-Yen Yeh. 2011. "On sampling type distribution from heterogeneous social networks." 111–122, Springer.

Li, Jiahui, Kun Kuang, Baoxiang Wang, et al. 2021. "Shapley Counterfactual Credits for Multi-Agent Reinforcement Learning." 934–942.

Li, Jia, Zhichao Han, Hong Cheng, Jiao Su, Pengyun Wang, Jianfeng Zhang, and Lujia Pan. 2019*a*. "Predicting Path Failure in Time-Evolving Graphs." 1279–1289.

Lin, Xu. 2010. "Identifying Peer Effects in Student Academic Achievement by Spatial Autoregressive Models with Group Unobservables." *Journal of Labor Economics*, 28(4): 825–860.

Lin, Xu. 2015. "Utilizing spatial autoregressive models to identify peer effects among adolescents." *Empirical Economics*, 49(3): 929–960.

Li, Pei-Zhen, Ling Huang, Chang-Dong Wang, and Jian-Huang Lai. 2019*b*. "EdMot: An Edge Enhancement Approach for Motif-aware Community Detection." *KDD '19*, 479–487.

Li, Rong-Hua, Jeffrey Xu Yu, Lu Qin, Rui Mao, and Tan Jin. 2015. "On random walk based graph sampling." 927–938, IEEE.

Liu, Meng, Youzhi Luo, Limei Wang, Yaochen Xie, Hao Yuan, Shurui Gui, Zhao Xu, Haiyang Yu, Jingtun Zhang, Yi Liu, Keqiang Yan, Bora Oztekin, Haoran Liu, Xuan Zhang, Cong Fu, and Shuiwang Ji. 2021*a*. "DIG: A Turnkey Library for Diving into Graph Deep Learning Research." *arXiv preprint arXiv:2103.12608*.

Liu, Xiaodong, Lung-fei Lee, and Christopher R. Bollinger. 2010. "An efficient GMM estimator of spatial autoregressive models." *Journal of Econometrics*, 159(2): 303 – 319.

Liu, Yifei, Chao Chen, Yazheng Liu, et al. 2020. "Shapley Values and Meta-Explanations for Probabilistic Graphical Model Inference." 945–954.

Liu, Zelei, Yuanyuan Chen, Han Yu, Yang Liu, and Lizhen Cui. 2021*b*. "GTG-Shapley: Efficient and Accurate Participant Contribution Evaluation in Federated Learning." *arXiv:2109.02053*.

Li, Yaguang, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting."

Li, Yongkun, Zhiyong Wu, Shuai Lin, Hong Xie, Min Lv, Yinlong Xu, and John CS Lui. 2019*c*. "Walking with Perception: Efficient Random Walk Sampling via Common Neighbor Awareness." 962–973, IEEE.

Li, Yujia, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. 2016. "Gated Graph Sequence Neural Networks."

Lomeli, Maria, Mark Rowland, Arthur Gretton, and Zoubin Ghahramani. 2019. "Antithetic and Monte Carlo Kernel Estimators for Partial Rankings." *Statistics and Computing*, 1127–1147.

DOI: 10.14754/CEU.2024.08

Lundberg, Scott M, and Su-In Lee. 2017. "A Unified Approach to Interpreting Model Predictions." 4768–4777.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning. 2015. "Effective Approaches to Attention-based Neural Machine Translation." 1412–1421.

Maiya, Arun S, and Tanya Y Berger-Wolf. 2010. "Sampling community structure." 701–710.

Maleki, Sasan, Long Tran-Thanh, Greg Hines, Talal Rahwan, and Alex Rogers. 2013. "Bounding the Estimation Error of Sampling-based Shapley Value Approximation." *arXiv:1306.4265*.

Maxwell, K.A. Friends. 2002. "The Role of Peer Influence Across Adolescent Risk Behaviors." *Journal of Youth and Adolescence*, 31: 267–277.

Maynard, Harold B, G J Stegemerten, and John L Schwab. 1948. *Methods-Time Measurement.* McGraw-Hill.

Mitchell, Rory, Joshua Cooper, Eibe Frank, and Geoffrey Holmes. 2021. "Sampling Permutations for Shapley Value Estimation." *arXiv:2104.12199*.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified Linear Units Improve Restricted Boltzmann Machines." 807–814.

Narayanan, Annamalai, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. "graph2vec: Learning Distributed Representations of Graphs."

Okhrati, Ramin, and Aldo Lipani. 2021. "A Multilinear Sampling Algorithm to Estimate Shapley Values." 7992–7999, IEEE.

Ortega, Francesc, and Giovanni Peri. 2013. "The effect of income and immigration policies on international migration." *Migration Studies*, 1(1): 47–74.

Ou, Mingdong, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. "Asymmetric transitivity preserving graph embedding." 1105–1114.

Owen, Guillermo. 1972. "Multilinear Extensions of Games." *Management Science*, 18(5-part-2): 64–79.

Owen, Guilliermo. 1977. "Values of Games with a Priori Unions." In *Mathematical Economics and Game Theory*. 76–88. Springer, Berlin, Heidelberg.

Panagopoulos, George, Giannis Nikolentzos, and Michalis Vazirgiannis. 2021. "Transfer Graph Neural Networks for Pandemic Forecasting."

Pareja, Aldo, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B Schardl, and Charles E Leiserson. 2020. "EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs." 5363–5370.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. "PyTorch: An imperative style, high-performance deep learning library." 8024–8035.

Patel, Roma, Marta Garnelo, Ian Gemp, et al. 2021. "Game-Theoretic Vocabulary Selection via the Shapley Value and Banzhaf Index." 2789–2798.

Pebesma, Edzer. 2017. "staRs: Spatiotemporal Arrays: Raster and Vector Datacubes." *https://github.com/r-spatial/stars*.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. "Scikit-learn: Machine learning in Python." *Journal of machine learning research*, 12(Oct): 2825–2830.

Peixoto, Tiago P. 2014. "The graph-tool python library." *figshare*.

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. 2014. "Deepwalk: Online learning of social representations." 701–710, ACM.

Perozzi, Bryan, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. "Don't Walk, Skip!: online learning of multi-scale network embeddings." 258–265, ACM.

Pintér, Miklós. 2011. "Regression games." *Annals of Operations Research*, 186(1): 263–274.

Prat-Pérez, Arnau, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. 2014. "High quality, scalable and parallel community detection for large real graphs." 225–236.

Qiu, Jiezhong, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec." 459–467, ACM.

Raghavan, Usha Nandini, Réka Albert, and Soundar Kumara. 2007. "Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks." *Physical review E*, 76(3): 036106.

Rehurek, Radim, and Petr Sojka. 2011. "Gensim—statistical semantics in python." *Retrieved from genism. org*.

Rey, Sergio J, and Luc Anselin. 2010. "PySAL: A Python Library of Spatial Analytical Methods." In *Handbook of Applied Spatial Analysis*. 175–193. Springer.

Rezvanian, Alireza, and Mohammad Reza Meybodi. 2015. "Sampling social networks using shortest paths." *Physica A: Statistical Mechanics and its Applications*, 424: 254–268.

Ribeiro, Bruno, and Don Towsley. 2010. "Estimating and sampling graphs with multidimensional random walks." 390–403.

Rob, Emanuele. 2020. "PySTAC: Python library for working with any SpatioTemporal Asset Catalog (STAC)." *https://github.com/stac-utils/pystac*.

Rozemberczki, Benedek, and Rik Sarkar. 2018. "Fast Sequence-Based Embedding with Diffusion Graphs." 99–107, Springer.

Rozemberczki, Benedek, and Rik Sarkar. 2020. "Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models." ACM.

Rozemberczki, Benedek, and Rik Sarkar. 2021. "The Shapley Value of Classifiers in Ensemble Games." 1558–1567.

Rozemberczki, Benedek, Carl Allen, and Rik Sarkar. 2019. "Multi-scale Attributed Node Embedding." *arXiv preprint arXiv:1909.13021*.

Rozemberczki, Benedek, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Olivér Kiss, Sebastian Nilsson, and Rik Sarkar. 2022. "The Shapley Value in Machine Learning." 5572–5579. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Rozemberczki, Benedek, Oliver Kiss, and Rik Sarkar. 2020*a*. "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs." ACM.

Rozemberczki, Benedek, Oliver Kiss, and Rik Sarkar. 2020*b*. "Little Ball of Fur: A Python Library for Graph Sampling." ACM.

Rozemberczki, Benedek, Paul Scherer, Oliver Kiss, Rik Sarkar, and Tamas Ferenci. 2021*a*. "Chickenpox Cases in Hungary: a Benchmark Dataset for Spatiotemporal Signal Processing with Graph Neural Networks."

Rozemberczki, Benedek, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar. 2021*b*. "PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models." *CIKM '21*, 4564–4573. New York, NY, USA:Association for Computing Machinery.

Rozemberczki, Benedek, Peter Englert, Amol Kapoor, Martin Blais, and Bryan Perozzi. 2020. "Pathfinder Discovery Networks for Neural Message Passing." *arXiv preprint arXiv:2010.12878*.

Rozemberczki, Benedek, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. "GEMSEC: Graph Embedding with Self Clustering." 65–72, ACM.

Rubinstein, Reuven Y, and Dirk P Kroese. 2016. *Simulation and the Monte Carlo Method.* John Wiley and Sons, Inc.

Sacerdote, Bruce. 2001. "Peer Effects with Random Assignment: Results for Dartmouth Roommates." *The Quarterly Journal of Economics*, 116(2): 681–704.

Sanders, Jason, and Edward Kandrot. 2010. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional.

Sarkar, Rik. 2011. "Low distortion delaunay embedding of trees in hyperbolic plane." 355–366, Springer.

Scherer, Paul, and Pietro Lio. 2020. "Learning Distributed Representations of Graphs with Geo2DR."

Schlichtkrull, Michael, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. "Modeling Relational Data with Graph Convolutional Networks." 593–607, Springer.

Seo, Youngjoo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. "Structured Sequence Modeling with Graph Convolutional Recurrent Networks." 362–373, Springer.

Shalev-Shwartz, Shai, and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms.* Cambridge university press.

Shapley, Lloyd. 1953. "A Value for N-Person Games." *Contributions to the Theory of Games*, 307–317.

Shi, Lei, Yifan Zhang, Jian Cheng, and Hanqing Lu. 2019. "Two-Stream Adaptive Graph Convolutional Networks for Skeleton-Based Action Recognition." 12026–12035.

Shim, Dongsub, Zheda Mai, Jihwan Jeong, Scott Sanner, et al. 2021. "Online Class-Incremental Continual Learning with Adversarial Shapley Value." Vol. 35, 9630–9638.

Singal, Raghav, George Michailidis, and Hoiyi Ng. 2021. "Flow-based Attribution in Graphical Models: A Recursive Shapley Approach." Vol. 139, 9733–9743.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *The Journal of Machine Learning Research*, 15(1): 1929–1958.

Staudt, Christian L, Aleksejs Sazonovs, and Henning Meyerhenke. 2016. "NetworKit: A Tool Suite for Large-Scale Complex Network Analysis." *Network Science*, 4(4): 508–530.

Stumpf, Michael PH, Carsten Wiuf, and Robert M May. 2005. "Subnets of scale-free networks are not scale-free: sampling properties of networks." *Proceedings of the National Academy of Sciences*, 102(12): 4221–4224.

Stutzbach, Daniel, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. 2008. "On unbiased sampling for unstructured peer-to-peer networks." *IEEE/ACM Transactions on Networking*, 17(2): 377–390.

Sun, Bing-Jie, Huawei Shen, Jinhua Gao, Wentao Ouyang, and Xueqi Cheng. 2017. "A non-negative symmetric encoder-decoder approach for community detection." 597–606, ACM.

Sundararajan, Mukund, and Amir Najmi. 2020. "The Many Shapley Values for Model Explanation." 9269–9278.

Sundararajan, Mukund, Kedar Dhamdhere, and Ashish Agarwal. 2020. "The Shapley Taylor Interaction Index." 9259–9268.

Sun, Dennis L, and Cedric Fevotte. 2014. "Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence." 6201–6205, IEEE.

Sun, Xin, Yanheng Liu, Jin Li, et al. 2012. "Feature Evaluation and Selection with Cooperative Game Theory." *Pattern recognition*, 45(8): 2992–3002.

Taheri, Aynaz, and Tanya Berger-Wolf. 2019. "Predictive Temporal Embedding of Dynamic Graphs." 57–64.

Taheri, Aynaz, Kevin Gimpel, and Tanya Berger-Wolf. 2019. "Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models." *WWW '19*, 301–307.

Tang, Jian, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. "Line: Large-scale information network embedding." 1067–1077.

Taylor, Paul, Christopher Harris, Thompson Comer, and Mark Harris. 2019. "CUDA-Accelerated GIS and Spatiotemporal Algorithms." *https://github.com/rapidsai/cuspatial*.

Touati, Sofiane, Mohammed Said Radjef, and SAIS Lakhdar. 2021. "A Bayesian Monte Carlo Method for Computing the Shapley Value: Application to Weighted Voting and Bin Packing Games." *Computers & Operations Research*, 125: 105094.

Tripathi, Sandhya, N Hemachandra, and Prashant Trivedi. 2020. "Interpretable Feature Subset Selection: A Shapley Value Based Approach." 5463–5472.

Tsitsulin, Anton, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. "Netlsd: hearing the shape of a graph." 2347–2356.

Tu, Cunchao, Yuan Yao, Zhengyan Zhang, Ganqu Cui, Hao Wang, Changxin Tian, Jie Zhou, and Cheng Yang. 2018. "OpenNE: An Open Source Toolkit for Network Embedding." *https://github.com/thunlp/OpenNE*.

Van Der Walt, Stefan, S Chris Colbert, and Gael Varoquaux. 2011. "The NumPy Array: a Structure for Efficient Numerical Computation." *Computing in science & engineering*, 13(2): 22–30.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention is All You Need." 6000–6010.

Verbeek, Kevin, and Subhash Suri. 2014. "Metric embedding, hyperbolic space, and social networks." 501–510.

DOI: 10.14754/CEU.2024.08

Verma, Saurabh, and Zhi-Li Zhang. 2017. "Hunt for the unique, stable, sparse and fast feature learning on graphs." 88–98.

Virtanen, Pauli, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2019. "SciPy 1.0–fundamental algorithms for scientific computing in Python." *arXiv preprint arXiv:1907.10121*.

Walt, Stéfan van der, S Chris Colbert, and Gael Varoquaux. 2011. "The NumPy array: a structure for efficient numerical computation." *Computing in Science & Engineering*, 13(2): 22–30.

Wang, Jianhong, Jinxin Wang, Yuan Zhang, Yunjie Gu, and Tae-Kyun Kim. 2021. "SHAQ: Incorporating Shapley Value Theory into Q-Learning for Multi-Agent Reinforcement Learning." *arXiv:2105.15013*.

Wang, Jiaxuan, Jenna Wiens, and Scott Lundberg. 2021. "Shapley Flow: A Graph-Based Approach to Interpreting Model Predictions." 721–729.

Wang, Tianhao, Johannes Rausch, Ce Zhang, et al. 2020. "A Principled Approach to Data Valuation for Federated Learning." In *Federated Learning*. 153–167. Springer Nature Switzerland.

Wang, Xiao, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. "Community Preserving Network Embedding." *AAAI'17*, 203–209.

Whitby, Michael A, Rich Fecher, and Chris Bennight. 2017. "GeoWave: Utilizing Distributed Key-Value Stores for Multidimensional Data." 105–122, Springer.

Williamson, Brian, and Jean Feng. 2020. "Efficient Nonparametric Statistical Inference on Population Feature Importance Using Shapley Values." 10282–10291.

Wilson, David Bruce. 1996. "Generating random spanning trees more quickly than the cover time." 296–303.

Winter, Eyal. 1989. "A Value for Cooperative Games with Levels Structure of Cooperation." *International Journal of Game Theory*, 18(2): 227–40.

Wu, Zonghan, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 2020. "Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks." 753–763.

Yanardag, Pinar, and S.V.N. Vishwanathan. 2015. "Deep Graph Kernels." 1365–1374.

Yang, Cheng-Lun, Perng-Hwa Kung, Chun-An Chen, and Shou-De Lin. 2013. "Semantically sampling in heterogeneous social networks." 181–182.

Yang, Cheng, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. 2017. "Fast network embedding enhancement via high order proximity approximation." 3894–3900.

DOI: 10.14754/CEU.2024.08

Yang, Cheng, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. "Network representation learning with rich text information."

Yang, Dingqi, Paolo Rosso, Bin Li, and Philippe Cudre-Mauroux. 2019. "NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching." 1162–1172.

Yang, Hong, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. "Binarized attributed network embedding." 1476–1481, IEEE.

Yang, Hongxia. 2019. "AliGraph: A Comprehensive Graph Neural Network Platform." 3165–3166.

Yang, Jaewon, and Jure Leskovec. 2013. "Overlapping community detection at scale: a non-negative matrix factorization approach." 587–596, ACM.

Yang, Shuang, and Bo Yang. 2018. "Enhanced Network Embedding with Text Information." 326–331, IEEE.

Yan, Tom, and Ariel Procaccia. 2021. "If You Like Shapley Then You Will Love the Core." *Proceedings of the AAAI Conference*, 5751–5759.

Ye, Fanghua, Chuan Chen, and Zibin Zheng. 2018. "Deep Autoencoder-like Nonnegative Matrix Factorization for Community Detection." *CIKM '18*, 1393–1402.

Yuan, Hao, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. 2021. "On Explainability of Graph Neural Networks via Subgraph Explorations." 12241–12252.

Yu, Bing, Haoteng Yin, and Zhanxing Zhu. 2018. "Spatio-Temporal Graph Convolutional Networks: a Deep Learning Framework for Traffic Forecasting." 3634–3640.

Yu, Le, Leilei Sun, Bowen Du, Chuanren Liu, Hui Xiong, and Weifeng Lv. 2020. "Predicting Temporal Sets with Deep Neural Networks." 1083–1091.

Zachary, Wayne W. 1977. "An information flow model for conflict and fission in small groups." *Journal of anthropological research*, 33(4): 452–473.

Zhang, Daokun, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. "SINE: Scalable Incomplete Network Embedding." 737–746, IEEE.

Zhang, Fan, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. 2020. "MediaPipe Hands: On-device Real-time Hand Tracking."

Zhang, Hao, Yichen Xie, Longjie Zheng, et al. 2021. "Interpreting Multivariate Shapley Interactions in DNNs." Vol. 35, 10877–10886.

Zhao, Ling, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2019. "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction." *IEEE Transactions on Intelligent Transportation Systems*, 21(9): 3848–3858.

DOI: 10.14754/CEU.2024.08

Zheng, Chuanpan, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020*a*. "GMAN: A Graph Multi-Attention Network for Traffic Prediction." Vol. 34, 1234–1241.

Zheng, Da, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. 2020*b*. "Learning Graph Neural Networks with Deep Graph Library." *WWW '20*, 305–306.

Zhou, Zhuojie, Nan Zhang, and Gautam Das. 2015. "Leveraging History for Faster Sampling of Online Social Networks." *Proceedings of the VLDB Endowment*, 8(10).

Zhu, Jiawei, Yujiao Song, Ling Zhao, and Haifeng Li. 2020. "A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting." *arXiv preprint arXiv:2006.11583*.

Zimányi, Esteban, Mahmoud Sakr, and Arthur Lesuisse. 2020. "MobilityDB: A Mobility Database Based on PostgreSQL and PostGIS." *ACM Transactions on Database Systems (TODS)*, 45(4): 1–42.